



LONG TERM STATISTICAL STUDIES OF IONOSPHERIC ABSORPTION

GEORGIOS NIKITAS
M.SC. DISSERTATION

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENT FOR THE DEGREE OF
MASTER IN SCIENCE IN SPACE COMMUNICATIONS ENGINEERING

SUPERVISORS

DR. FARIDEH HONARY
DR. STEVE MARPLE

LANCASTER UNIVERSITY
DEPARTMENT OF COMMUNICATION SYSTEMS
2000

ACKNOWLEDGEMENTS

Special thanks to Dr. F. Honary for the project supervision and for providing valuable help on major theoretical parts. I am grateful to Dr. S. Marple for the significant help on the MATLAB programming and for the introduction on the IRIS Toolkit. Moreover, many thanks to the imaging riometer project (IRIS) at Kilpisjärvi (which is a joint project between the Department of Communications Systems of the University of Lancaster, funded by the UK Particle Physics and Astronomy Research Council, and the Sodankylä Geophysical Observatory) for supplying us all the available Data for the years October 1994 to June 2000. I also acknowledge the rest of the research staff of the Ionosphere and Radio Propagation Group (Department of Communications Systems, Lancaster University) for any additional assistance on the project.

I would like to thank my girlfriend Katerina for providing ideas and comments on many sections of this project and finally I must thank my family Michael, Stella and Matina for their considerable support during my academic years.

ABSTRACT

An intelligent, user friendly and powerful toolkit has been designed and implemented in MATLAB to enable statistical analysis on IRIS data. This toolkit provides a catalogue of analysed IRIS data and statistical analysis in automated and efficient manner. The end results are histograms, tables and plots that are generated automatically from the output of the statistical analysis. Although this toolkit is written for IRIS data, it can easily be adopted for any other data sets. The only necessary modification is to implement a different set of rules that provides the catalogue. For any large data set; it is of vital importance to have such an advanced software to exploit the data fully.

LIST OF SYMBOLS

IRIS	Imaging Riometer for Ionospheric Studies
SGO	Sodankylä Geophysical Observatory
PPARC	Particle Physics and Astronomy Research Council
AA	Auroral Absorption
PCA	Polar Cap Absorption

LIST OF FIGURES

Figure 2.1: The Kilpisjarvi IRIS beams.	4
Figure 3.1: Data Analysis Structure Diagram.	11
Figure 3.2: Data Base Structure Diagram.	12
Figure 3.3: Statistical Analysis Structure Diagram.	13
Figure 3.4: Project Design.	17
Figure 4.1: Contents of the pca_1995_4and10.dat file.	30
Figure 4.2a & 4.2b: Event occurred on the 1 st of March 1995 at 11:39:00.	32
Figure 4.3: Statistical Analysis I, March 1995 (wide beam).	35
Figure 4.4: Statistical Analysis II, March 1995 (wide beam).	38
Figure 4.5: Statistical Analysis III, All Events, Year 1995 (wide beam).	41
Figure 4.6: Statistical Analysis III, All Events, Year 1995 (wide beam).	41
Figure 5.4.1: Sample of the Statistical Analysis Results – Type I, January 1999.	50
Figure 5.4.2: Sample of the Statistical Analysis Results – Type II, January 1999.	52
Figure 5.4.3: Sample of the Statistical Analysis Results – Type III, January 1999.	55
Figure 5.1.1-5.1.69: Statistical Analysis I Results Sep1994-Jun2000.	<i>APPENDIX C</i>
Figure 5.2.1-5.2.69: Statistical Analysis II Results Sep1994-Jun2000.	<i>APPENDIX C</i>
Figure 5.3.1-5.3.7: Statistical Analysis III Results Sep1994-Jun2000.	<i>APPENDIX C</i>
Figure 5.3.1b-5.3.7b: Statistical Analysis III Results Sep1994-Jun2000.	<i>APPENDIX C</i>
Figure 5.3.1c-1-5.3.7c-1: Statistical Analysis III Results Auroral Absorptions Sep1994-Jun2000.	<i>APPENDIX C</i>
Figure 5.3.1c-2-5.3.7c-2: Statistical Analysis III Results Spike Events Sep1994-Jun2000.	<i>APPENDIX C</i>
Figure 5.3.1c-3-5.3.7c-3: Statistical Analysis III Results Polar Cap Absorptions Sep1994-Jun2000.	<i>APPENDIX C</i>

Figure 5.3.1c-4-5.3.7c-4: Statistical Analysis III Results Noise
(Interference) Sep1994-Jun2000.

APPENDIX C

Figure 5.3.1c-5-5.3.7c-5: Statistical Analysis III Results Data Gaps
(Missing Data) Sep1994-Jun2000.

APPENDIX C

LIST OF TABLES

Table 1.1: Project Timetable.	2
Table 4.1: Project Functions & Files.	18
Table 4.2: Orientation of Information into the Pseudo Results File.	21
Table 4.3: Contents of the <i>zpseudo19951012_19960223.gn</i> file.	23
Table 4.4: Orientation of Information into the Filter Results File.	24
Table 4.5a: Contents of the <i>zpseudo19951012_19960223.gn</i> file.	25
Table 4.5b: Contents of the <i>zfilter19951012_19960223.g</i> file.	26
Table 4.6: Histogram Statistical Analysis I, March 1995 (widebeam).	36
Table 4.7: Histogram Statistical Analysis II, March 1995 (widebeam).	38
Table 4.8: Histogram Statistical Analysis III, March 1995 (widebeam).	42
Table 4.9: Characteristics of the Auroral Absorptions.	44
Table 4.10: Pseudo Code for the Auroral Absorptions.	44
Table 4.11: Characteristics of the Spike Events.	44
Table 4.12: Pseudo Code for the Spike Events.	44
Table 4.13: Characteristics of the Polar Cap Absorptions.	45
Table 4.14: Pseudo Code for the Polar Cap Absorptions.	45
Table 4.15: Characteristics of the Noise Events.	45
Table 4.16: Pseudo Code for the Noise Events.	46
Table 4.17: Characteristics of the Data Gaps.	46
Table 4.18: Pseudo Code for the Data Gaps.	46
Table 5.4.1: Sample of the Statistical Analysis Results – Type I, Year 1999.	50
Table 5.4.2: Sample of the Statistical Analysis Results – Type II, Year 1999.	52
Table 5.4.3: Sample of the Statistical Analysis Results – Type III,	54

Year 1999.

Table 6.4.1: Average Duration for PCA events on each Month 56

(Years 1994-2000).

Table 6.4.2: Duration Distribution for PCA events on each Month 57

(Years 1994-2000).

Table 7.1.1: Total/Average Number of AAs on each month 59

(hours 00-04).

Table 7.1.2: Total/Average Number of AAs on each month 60

(hours 04-08).

Table 7.1.3: Total/Average Number of AAs on each month 61

(hours 08-12).

Table 7.1.4: Total/Average Number of AAs on each month 62

(hours 12-16).

Table 7.1.5: Total/Average Number of AAs on each month 63

(hours 16-20).

Table 7.1.6: Total/Average Number of AAs on each month 64

(hours 20-24).

Table 7.2.1: Total/Average Number of AAs on each month 65

(between 2-6 minutes).

Table 7.2.2: Total/Average Number of AAs on each month 66

(between 6-30 minutes).

Table 7.2.3: Total/Average Number of AAs on each month 67

(between 30mins-1hour).

Table 7.2.4: Total/Average Number of AAs on each month 68

(between 1-2 hours).

Table 7.2.5: Total/Average Number of AAs on each month 69

(between 2-3 hours).

Table 7.2.6: Total/Average Number of AAs on each month 70

(between 3-5 hours).

Table 5.1.1-5.1.7: Statistical Analysis I, Summary Results Table *APPENDIX B*

Year 1994-2000.

Table 5.2.1-5.2.7: Statistical Analysis II, Summary Results Table *APPENDIX B*

Year 1994-2000.

Table 5.3.1-5.3.7: Statistical Analysis III, Summary Results
Table Year 1994-2000.

APPENDIX B

CONTENTS

ACKNOWLEDGEMENTS	i
ABSTRACT	ii
LIST OF SYMBOLS	iii
LIST OF FIGURES	iv
LIST OF TABLES	vi
CONTENTS	ix

CHAPTER 1. INTRODUCTION	1
--------------------------------	----------

CHAPTER 2. THEORETICAL BACKGROUND	3
--	----------

2.1 INTRODUCTION	3
2.2 IMAGING RIOMETERS AND IRIS	3
2.3 ABSORPTION INCIDENCES	4
2.3.1 AURORAL ABSORPTION	4
2.3.2 SUBSTORM	4
2.3.3 POLAR CAP ABSORPTION	5
2.4 NOISE INCIDENCES	5
2.4.1 MAN-MADE INTERFERENCE	5
2.4.2 SOLAR EMISSION	6
2.4.3 SCINTILLATION	6
2.5 OTHER INCIDENCES	7
2.5.1 DATA GAP	7
2.6 CONCLUSION	7

CHAPTER 3. PROJECT DESIGN **8**

3.1 INTRODUCTION	8
3.2 FUNCTIONS STRUCTURE	8
3.2.1 DATA ANALYSIS DESIGN	8
3.2.2 DATA BASE DESIGN	11
3.2.3 STATISTICAL ANALYSIS DESIGN	12
3.3 IRISTOOL v2	14
3.3.1 IRIS TOOLKIT CLASSES	14
3.4 CONCLUSION	17

CHAPTER 4. PROJECT IMPLEMENTATION **18**

4.1 INTRODUCTION	18
4.2 FUNCTIONS GUIDE	20
4.2.1 DATA ANALYSIS	20
<i>IRISDATA_PSEUDOANALYSIS.M</i>	20
<i>ABS_FILTERANALYSIS.M</i>	23
<i>CREATECATALOGUE.M</i>	26
<i>IRISDATA_LOAD.M</i>	27
<i>SCRIPTANALYSIS.M</i>	27
4.2.2 DATA BASE	28
<i>DBASEIRIS.M</i>	28
<i>SHOWEVENT.M</i>	31
4.2.3 STATISTICAL ANALYSIS	34
<i>STATAA_TOD.M</i>	34
<i>STATAA_DUR.M</i>	37
<i>STATMULTILONG.M</i>	39
4.3 FUNCTIONS ANALYSIS	43
4.3.1 EVENTS' RULES	43

<i>AURORAL ABSORPTION</i>	43
<i>SPIKE EVENT</i>	44
<i>POLAR CAP ABSORPTION</i>	45
<i>NOISE</i>	45
<i>DATA GAP</i>	46
4.3.2 EVENTS' MEAN VALUE	46
4.3.3 OTHER ALGORITHMS	48
4.4 CONCLUSION	48
 CHAPTER 5. PROJECT STATISTICAL RESULTS	 49
5.1 LONG TERM STATISTICS – TYPE I	49
5.2 LONG TERM STATISTICS – TYPE II	51
5.3 LONG TERM STATISTICS – TYPE III	53
 CHAPTER 6. PCAs DURATION DISTRIBUTION	 56
 CHAPTER 7. AURORAL ABSORPTIONS DISTRIBUTION	 58
7.1 AURORAL ABSORPTION EVENTS' TIME OF DAY DISTRIBUTION	58
7.2 AURORAL ABSORPTION EVENTS' DURATION DISTRIBUTION	65
 CHAPTER 8. PROJECT LIMITATIONS & FURTHER WORK	 71
8.1 LIMITATIONS	71
8.2 FURTHER WORK	72
 CHAPTER 9. CONCLUSIONS	 74

REFERENCES	77
-------------------	-----------

APPENDICES	80
-------------------	-----------

APPENDIX A – MATLAB FUNCTIONS	-
APPENDIX B – STATISTICAL ANALYSIS RESULTS (TABLES)	-
APPENDIX C – STATISTICAL ANALYSIS RESULTS (HISTOGRAMS)	-

CHAPTER 1. INTRODUCTION

There are different types of absorptions, as well as noise and data gaps within the IRIS data. The IRIS is now operating for over six years and statistical studies can now be performed on this large volume of data.

This project is of great importance since it enables the study of different absorption events on a statistical basis. In particular, the aim of this project is to design and implement in MATLAB a toolkit that will enable any user to:

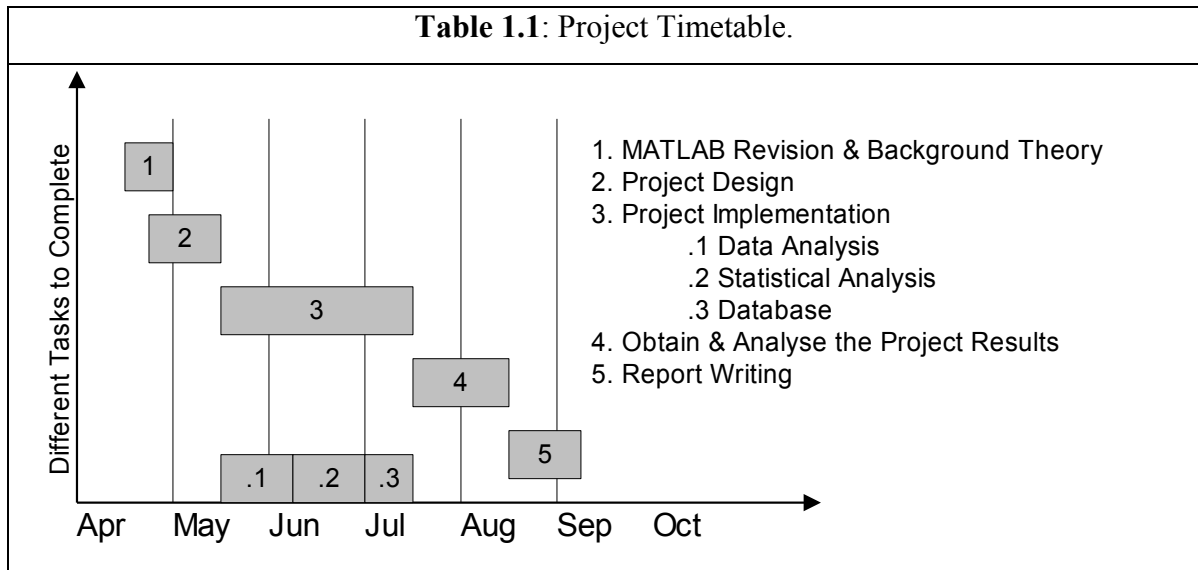
- i. Load a large amount of the IRIS data (e.g. load the IRIS data for the year 1997) from the IRIS server in a fast and efficient way, so that the server's memory will not overload.
- ii. Perform data analysis (data recognition) by a set of rules (i.e. auroral absorption, spike event, polar cap absorption, noise and data gap recognition) and to save the results in files whose name is set automatically by the program.
- iii. Create an entire catalogue containing the results of the analysed data for the years 1994-2000.
- iv. Produce histograms and tables of the statistical analysis.
- v. Produce a database server equipped with a search engine in order to collect detailed information for all the events and for any date and time (i.e. return for each separate event its start and end date and time, its duration, its max and mean values and a summary of the search analysis as well).

In this report a detailed study of polar cap absorption analysis is presented, where the whole IRIS data set for the years 1994-2000 has been searched for this event and the distribution and characteristics of this type of absorption is reported. Furthermore, the monthly and yearly distribution for the number of the auroral absorptions for different durations and different times of their occurrence is presented.

Note that all the project specifications that have been stated and discussed above have been designed, implemented and tested successfully and their thorough analysis follows on the next chapters.

The next chapter (chapter 2) of this report discusses briefly the IRIS system and a brief theory describing different types of absorption. The toolkit's structure, the project design and the MATLAB functions structure are reported on chapter 3. Chapter 4 explains how to use easily and efficiently the toolkit's functions. Also all possible inputs that are generated by the toolkit's functions are discussed. Chapter 5 illustrates the results produced by the statistical analyses. The distribution of the duration of the polar cap absorptions as well as the distribution of the auroral absorptions is presented in chapters 6 and 7 respectively. Finally, chapter 8 discusses the project limitations and suggests any further work. The conclusions are given on chapter 9. The source code for the toolkit's MATLAB functions is provided in Appendix A. Since there are a number of plots and tables representing the statistical analysis for the years 1994 to 2000, only a sample of results are presented in chapter 5 and the rest are included as Appendix B and C.

The bar chart on the Table 1.1 gives an approximate timetable for the work that has been done on the project.



CHAPTER 2. THEORETICAL BACKGROUND

2.1 INTRODUCTION

It is necessary to make a brief revision on the IRIS system (e.g. what exactly is the IRIS systems, its location and generally what operations does provides), because in our project the data that is observed by the IRIS system will be used on the Data Analysis part. Further, in order to understand how to design the different absorption and noise events and in order to be able to explain the results that will be obtained later from the Statistical Analysis part, we should understand first the theory and the characteristics of all those events.

2.2 IMAGING RIOMETERS AND I.R.I.S.

The Kilpisjärvi IRIS system in northern Finland (69.05° N, 20.79° E) is supervised by Lancaster University (UK) and operated in conjunction with Sodankylä Geophysical Observatory (SGO), Finland. It has been in operation since 2nd September 1994. IRIS is one of the **UK STP national facilities** funded by **PPARC**. The Figure 2.1 shows the projection onto the ionosphere at 90 km altitude of the Kilpisjärvi IRIS beams.

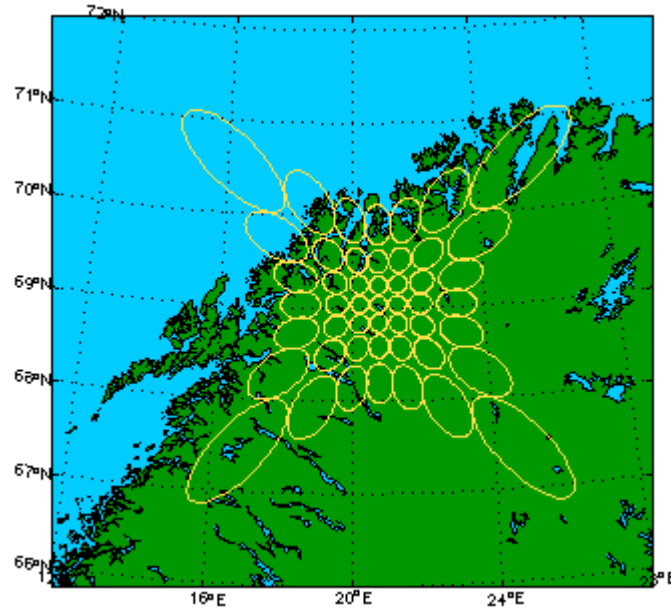


Figure 2.1: The Kilpisjärvi IRIS beams.

The system operates at 38.2 MHz and produces an array of 49 narrow beams with widths between 13° and 16° . The phasing of the array is achieved using an assembly of Butler matrices. The basic scanning interval of the array is one second.

2.3 ABSORPTION INCIDENCES

2.3.1 AURORAL ABSORPTION

Under geomagnetic disturbed conditions energetic electrons can enter the D-region at high latitudes and produce additional ionisation, which causes absorption. This is the Auroral Absorption (AA), which occurs mostly in the auroral regions.

2.3.2 SUBSTORM

During substorms, in the Polar Regions, aurora becomes widespread and intense, also much more agitated, and the Earth's magnetic field is disturbed. Out in space, ions and electrons flow in much greater numbers and at higher energies, and changes in the magnetic field are much more profound than those seen at Earth.

Typically, a storm takes about half a day to develop, and it gradually decays over the next few days.

Magnetic storms are relatively rare. On the other hand, smaller "**substorms**" observable mainly in Polar Regions (and in space!) present a clearer pattern and seem to be more fundamental. They are also much more frequent, often just hours apart. The two are of course related, and during magnetic storms intense substorms are generally observed in the Polar Regions. "Storms" distinguish themselves by injecting appreciable numbers of ions and electrons from the tail into the outer radiation belt, and their worldwide magnetic disturbance reflects a rapid growth of the ring current. Substorms usually do not inject as many particles. It might thus be that magnetic storms are merely sequences of very intense substorms, but additional factors are also involved--in particular, magnetic storms require external stimuli such as the arrival of a shock front or a fast stream in the solar wind.

On Earth the most visible sign of a substorm is **a great increase of polar auroras** in the midnight auroral zone. At ordinary times, quiescent auroral arcs are often seen there, but following the onset of a substorm, they intensify, move rapidly (mostly poleward) and expand, until they may cover much of the sky. Their activity may build up for half an hour and then decay, but as with atmospheric weather, patterns are quite variable.

2.3.3 POLAR CAP ABSORPTION

Strong D-region Ionisation resulting from an influx of energetic protons, usually after a major solar flare, produces PCA. The event lasts most of the day (pausing only during the middle of the night). PCA events often last several days.

2.4 NOISE INCIDENCES

2.4.1 MAN-MADE INTERFERENCE

Since we are measuring very low signal powers, Man-Made Interference can be a problem. Interference levels can be high in the HF range due to Ionospheric propagation. An important type of Man-Made Interference (which occurs near Kilpisjarvi, northern Finland) is the existence of a Heater (High Power HF installation). The location of the Heater is known; therefore only 1 beam and 2

neighbouring beams can be affected. At the IRIS system, they keep a file of all the times the Heater is ON, to avoid confusions with the other events.

2.4.2 SOLAR EMISSION

Until now, solar γ -ray emissions have only been detected during solar flares. However, there are several scenarios (e.g., microflares) in which γ -ray emission might be detectable when there is no significant solar activity. These processes would be related to the general issue of small-scale energy releases and the general problem of solar coronal heating. Such emissions might also be observed to vary as a function of solar cycle.

2.4.3 SCINTILLATION

Signal variation due to naturally occurring (or sometimes man-made) irregularities in the ionosphere. The effect is the same as the visible twinkling of stars due to variations in the atmosphere. Scintillation is readily identified in the data. Scintillation is most easily explained by analogy to a diffraction grating. Ionospheric irregularities are the equivalent of the diffraction grating. The received signal is then the sum of signal from multiple paths. As the path lengths change the signal varies due to constructive and destructive interference, hence the star twinkles.

Ionospheric scintillation is characterised by large variations in the received power. It is not often seen on the wide beam antenna, neither at lower time resolutions (such as at 120 seconds). This is because wide beams and long sampling intervals both average out the variation in cosmic noise level. Any scintillation seen in the wide beam antenna is a clue that the ionosphere was very irregular at that time.

At Kilpisjärvi scintillation is only apparent when Cassiopeia (or occasionally Cygnus) are in view.

2.5 OTHER INCIDENCES

2.5.1 DATA GAP

We cannot predict the time it will occur, neither for how long and has no magnitude (NaN → Not a Number). The power supply of the IRIS system is responsible for this phenomenon; e.g. when the system goes off (probably because of a failure).

Note that the IEEE Standard 754 defines a class of numbers known as NaN, or Not a Number. This value is used by the IRIS software to indicate missing data.

2.6 CONCLUSION

In order to be able to continue on the project's design and implementation stages, a brief explanation of the IRIS system and the characteristics of some various absorption and noise incidences had to be discussed first.

CHAPTER 3. PROJECT DESIGN

3.1 INTRODUCTION

Considering the project specifications, our functions have been implemented in MATLAB and are divided into three main groups: a) the Data Analysis group, b) the Data Base group and c) the Statistical Analysis group. The project is designed such that the three groups together are dependent on each other; i.e. the outputs of the one group of functions are the inputs of the other one or two groups of functions and vice versa. Further, each group of functions is designed such that each function to perform only one or at least a similar group of actions, to be as simple as possible and finally all the functions to be interlocked.

The current IRIS Toolkit (Iristool v2) is a complete software package, which has been designed in MATLAB programming and some of its functions will be necessary for us to use. Therefore in order to design the project efficiently and next to continue on its implementation stage, it should be clear to us the structure of the Iristool v2, the functions that consist of, what each function accomplishes and how they do operate.

3.2 FUNCTIONS STRUCTURE

3.2.1 DATA ANALYSIS DESIGN

Initially, we have been asked to design a group of function that can analyse the iris data and save the results in files of ASCII format.

But, in order to be able to perform data analysis on the iris data, first we have to load the iris data from the iris server. For this operation, the function *irisdata_load.m* loads the requested iris data for a specific start and end times, resolution, location and number of beams and returns in matrices the data numbers and many other useful information as well.

Next, a function will analyse the loaded iris data and save the results of the data analysis in a file. One solution was to use a function that would load the requested iris data and next analyse it; but if a user request to load and analyse a big

amount of the iris data (e.g. one month or more), then this action would cause memory and processing problems to the iris server, the whole operation would become too slow and there would be a big possibility of crashing the process. Therefore, in order to avoid slowing down the server and analysing the requested iris data faster, it has been decided to load the iris data in smaller periods of time. Also considering the case that the iris data is stored into the iris server in objects and each object is known that contains data of one hour; it has been decided each time to load one hour of iris data and then analyse it and save the results in a file. This action will provide us the faster loading time of the iris data.

Concluding, the function *irisdata_pseudoanalysis.m* has been implemented, which loads one hour of the iris data (i.e. calls the *irisdata_load.m* function), next applies data analysis on the iris data (i.e. searches for a. the various events that can be seen from the iris system: auroral absorptions, spike events, polar cap absorptions, noise and data gaps, b. the beam or the beams that have been used by the iris system to collect the data and c. the start time, end time, duration, max value and mean value of those events) and then saves the results into a file (*zpseudoStartYearMonthDay_EndYearMonthDay.gn*). Inside the function, a loop is used to repeat the above process for one month and saves the analysis results into the same file. Finally, a file will be composed (*zpseudoStartYearMonthDay_EndYearMonthDay.gn*) and will contain the results of the data analysis of the requested month of a year. At the end, running this function for all the six years (from September 1994 until June 2000) of the available iris data, we create a folder (*pseudo results*) that contains the results files of the data analysis for every month of these six years.

Having solved the problem for loading the iris data, a second major problem is arisen after completing the data analysis process. The function *irisdata_pseudoanalysis.m* provides: a) correct results only for the events that start and end on the specific hour of the loaded iris data and b) incorrect results when the events start on a specific hour of the loaded iris data and end on a different hour of the loaded iris data. In order to understand the problem, an example is given. Let an event happens on the 11th of October 1995 14:52:00 and has a duration of 1 hours and 23 minutes (i.e. the start time is at 14:52:00 and the end time is at 16:15:00). Running the *irisdata_pseudoanalysis.m* function, one hour of iris data is loaded (i.e. the 14th hour of the 11th of October 1995) and next this hour of data is analysed. Therefore an event

will be recognised and saved in a file, with a start time 14:52:00 and end time 14:59:00. Continuing the process, the next hour of the iris data is loaded (i.e. the 15th hour of the 11th of October 1995) and next this hour of data is analysed. Therefore a second event will be recognised and saved in the same file, with a start time 15:00:00 and end time 15:59:00. Continuing further the process, the next hour of the iris data is loaded (i.e. the 16th hour of the 11th of October 1995) and next this hour of data is analysed. Therefore a third event will be recognised and saved in the same file, with a start time 16:00:00 and end time 16:15:00.

The solution of the last problem is given with the new function (*abs_filteranalysis.m*). This function searches the results file (*zpseudoStartYearMonthDay_EndYearMonthDay.gn*) of the previous data analysis for the start time of the first incorrect event and the end time of the last incorrect event and combine the simultaneous incorrect events into a one single (correct) event and saves it in a new results file (*zfilterStartYearMonthDay_EndYearMonthDay.gn*). The correct events from the *zpseudoStartYearMonthDay_EndYearMonthDay.gn* file are simply saved into the *zfilterStartYearMonthDay_EndYearMonthDay.gn* file. Therefore, running the *abs_filteranalysis.m* function for all the six years (from September 1994 until June 2000) of the available (*pseudo results*) folder, we create a new folder (*filter results*) that contains the results files of the new (filter) data analysis for every month of these six years.

Having designed the biggest part of the Data Analysis group, one extra function has been designed (*createcatalogue.m*) to collect the results of the data analysis from the (*filter results*) folder for all the months of one single year and save them in a file called (*zcatalogueYear.gn*). After running this function for six times (i.e. one time for every year), we create a new folder (*catalogue results*), which contains the results files for every year of these the six years. Note that the folders (*filter results*) and (*catalogue results*) are actually contain exactly the same information, with the only difference that on the (*filter results*) folder the results are oriented for every month of a year of all the six years and on the (*catalogue results*) folder the results are oriented for every year of all the six years. The reason of collecting the results of the data analysis in a two different forms (folders) is that: in the Data Base and Statistical Analysis stages, it is easier to use sometimes the results from the (*filter results*) folder and sometimes times from the (*catalogue results*) folder.

Finally, a script function (*scriptanalysis.m*) calls the *irisdata_pseudoanalysis.m* and *abs_filteranalysis.m* functions and with a loop it applies the loading and the data analysis processes for one whole year of the iris data. After running the above script for six times, it completes the data analysis for all the six years of the available iris data.

The Figure3.1 lists the functions structure for the Data Analysis group.

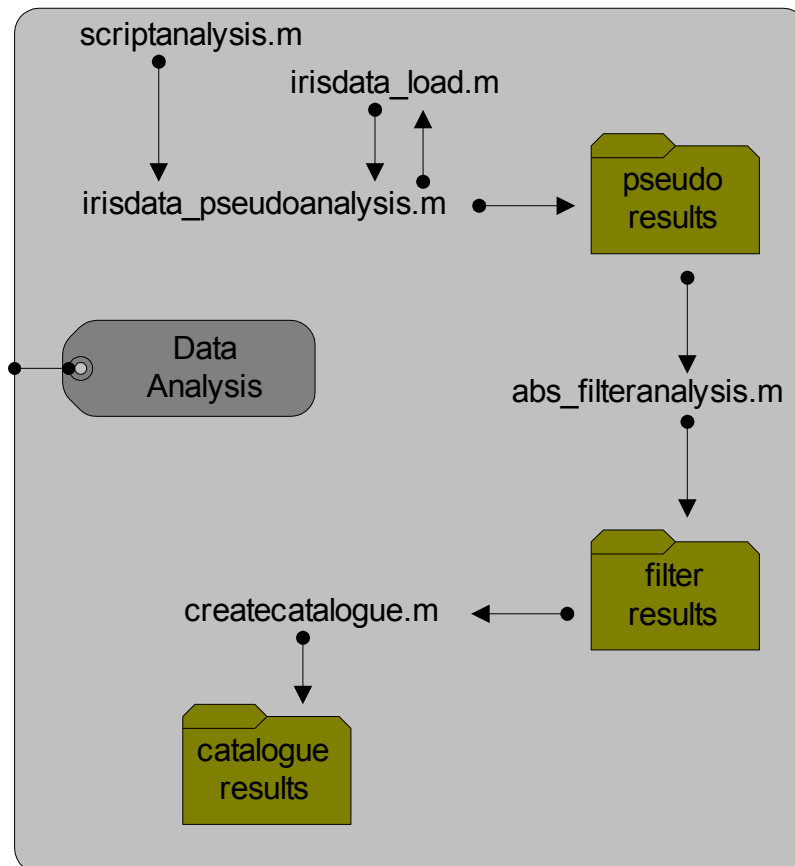


Figure 3.1: Data Analysis Structure Diagram.

3.2.2 DATA BASE DESIGN

The second group of functions is the Data Base group. The design of this system is quite simple; only one function (*dbaseiris.m*) is enough to fulfil the requirements of a Data Base system. The function needs to be very flexible, because the user must have the ability to request many different operations from the (*dbaseiris.m*) function (i.e. a Data Base system should provide to the user many different actions) e.g. search for a specific type of events (AA, PCA, Noise etc.) for a specific period of a year (e.g. September 1998) and of course the user should be able to save the results of the analysis in a file. Note also that there is a possibility the user

will not enter all the necessary information to the function, in that case the function searches for all the possible combinations of the missed input (e.g. if the user will provide only the year and not the months, the function searches for all the 12 months; or if the user will not specify the name of the file, then the function saves the results in a default file; similarly for the type of events. Note that the inputs for the Data Base system are taken from the (*catalogue results*) folder from the Data Analysis design.

An extra function (*showevent.m*) of the Data Base system is used to plot a specific event in an x-y axis (i.e. Absorption vs. Time) and to display a group of images, exactly as the event has been seen by the IRIS system. The user enters only the start time of the event and the function searches on the Data Analysis results files to find the requested event. The inputs for the (*showevent.m*) function are taken from the (*filter results*) folder from the Data Analysis design.

The design of the Data Base system is shown in Figure 3.2.

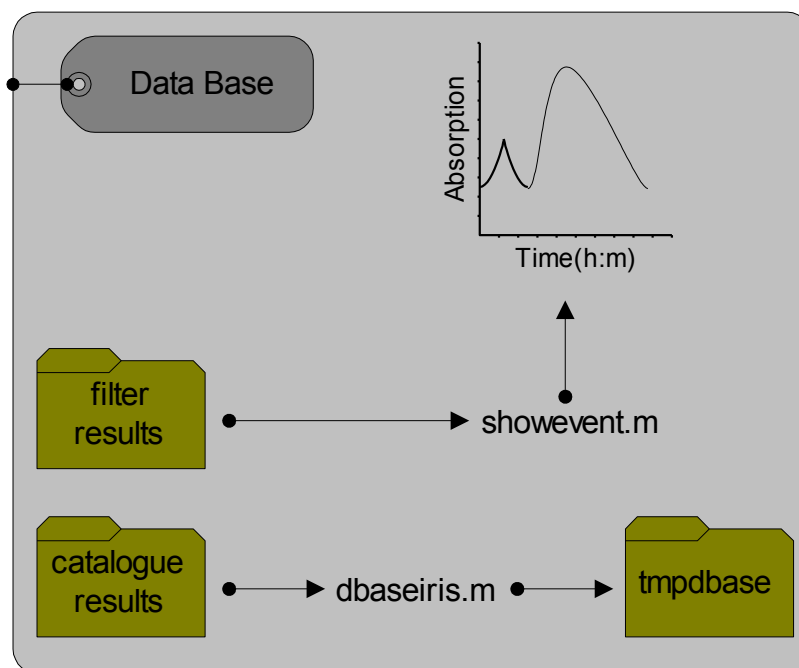


Figure 3.2: Data Base Structure Diagram.

3.2.3 STATISTICAL ANALYSIS DESIGN

The final and most important part of our design is the Statistical Analysis group of functions. Three different types of functions for the Statistical Analysis have been produced. Once the results of the Data Analysis stage have been produced (i.e.

the folders (*filter results*) and (*catalogue results*) are completed), then it is possible to start running the Statistical Analysis group of functions.

The first type of the Statistical Analysis group, the function (*stataa_tod.m*) reads the contents of the (*filter results*) folder and produces one or more histograms; and displays on an x-y axis the number of auroral absorptions and the number of spike events, depending on the time of their occurrence (for one or more months of a year).

Similarly the second type, the function (*stataa_dur.m*) reads the contents of the (*filter results*) folder and produces one or more histograms; and displays on an x-y axis the number of auroral absorptions, depending on their duration (for one or more months of a year).

Finally the third type, the function (*statmultilong.m*) reads the contents of the (*catalogue results*) folder and produces (depending on the user's request) one or more histograms; and displays on an x-y axis the number of events (auroral absorptions or spike events or polar cap absorptions or noise events or data gaps or all the events together), depending on the month of a year that they have occurred for a whole year.

The Figure3.3 lists the functions diagram for the Statistical Analysis group.

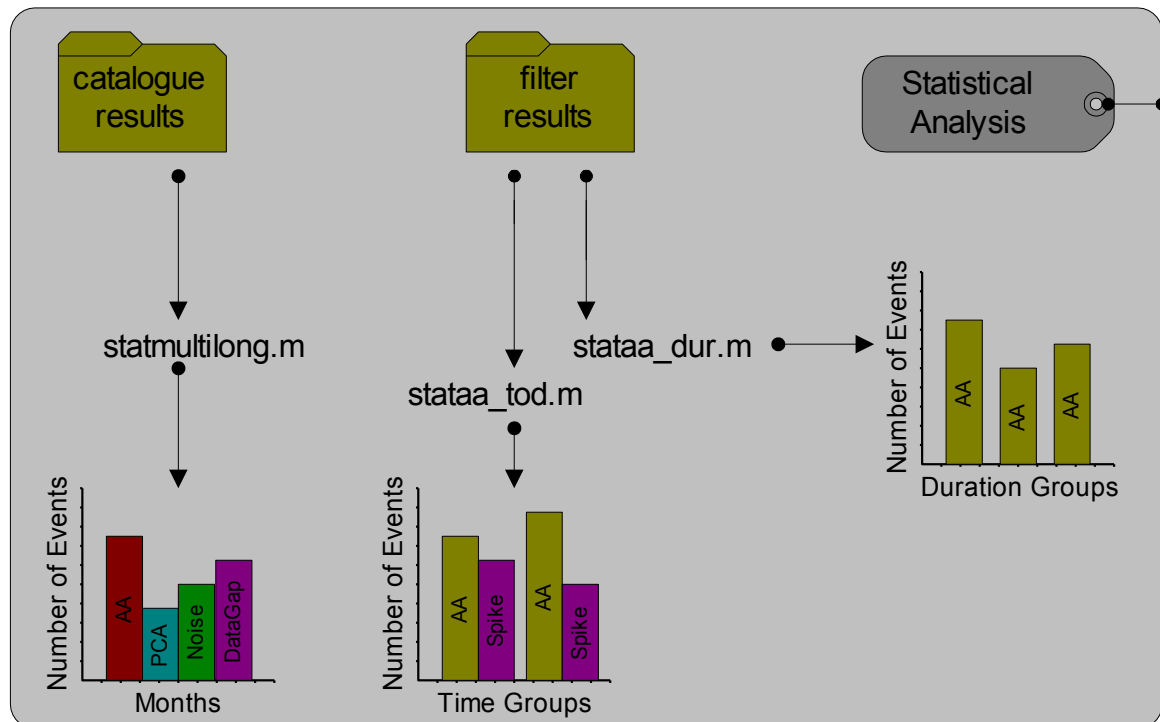


Figure 3.3: Statistical Analysis Structure Diagram.

3.3 IRISTOOL V2

As it has been mentioned earlier, the iristool v2 consists of various functions. Many of the iristool v2 function are very useful for our project and therefore in order to use them efficiently, it is necessary to understand the main structure of the IRIS Toolkit.

All the information given in that section has been found from the following web page: <http://www.dcs.lancs.ac.uk/iono/iris/iristool/classes/>.

3.3.1 IRIS TOOLKIT CLASSES

The classes have been split into 4 parts, basic, data, GUI and filter classes. For full information you should consult the relevant MATLAB documentation (<http://www.dcs.lancs.ac.uk/marple/cgi-bin/wmathelp>).

Basic Classes

time
dv

timespan
days seconds negative

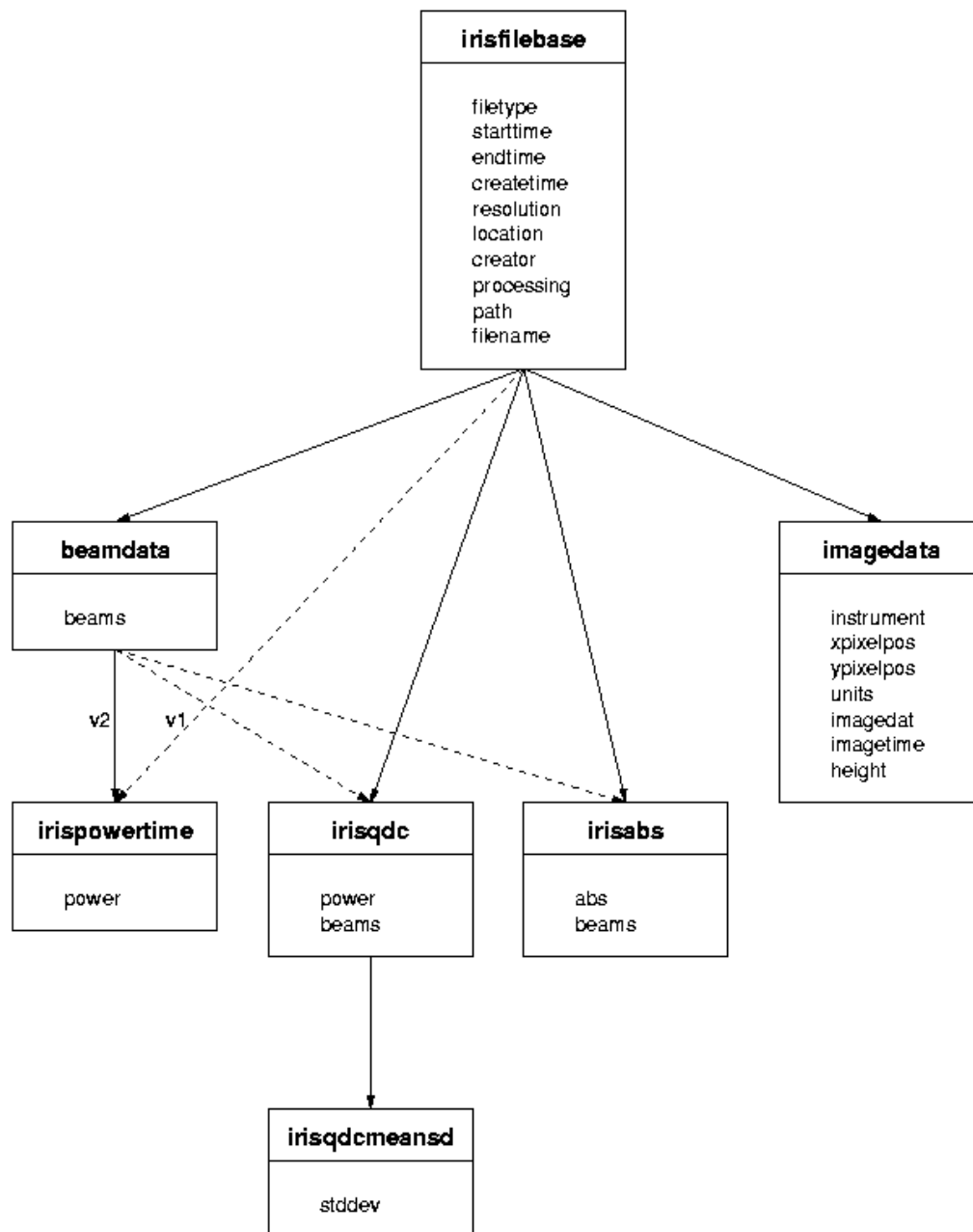
irisint16
i16

irisfiletype
name version

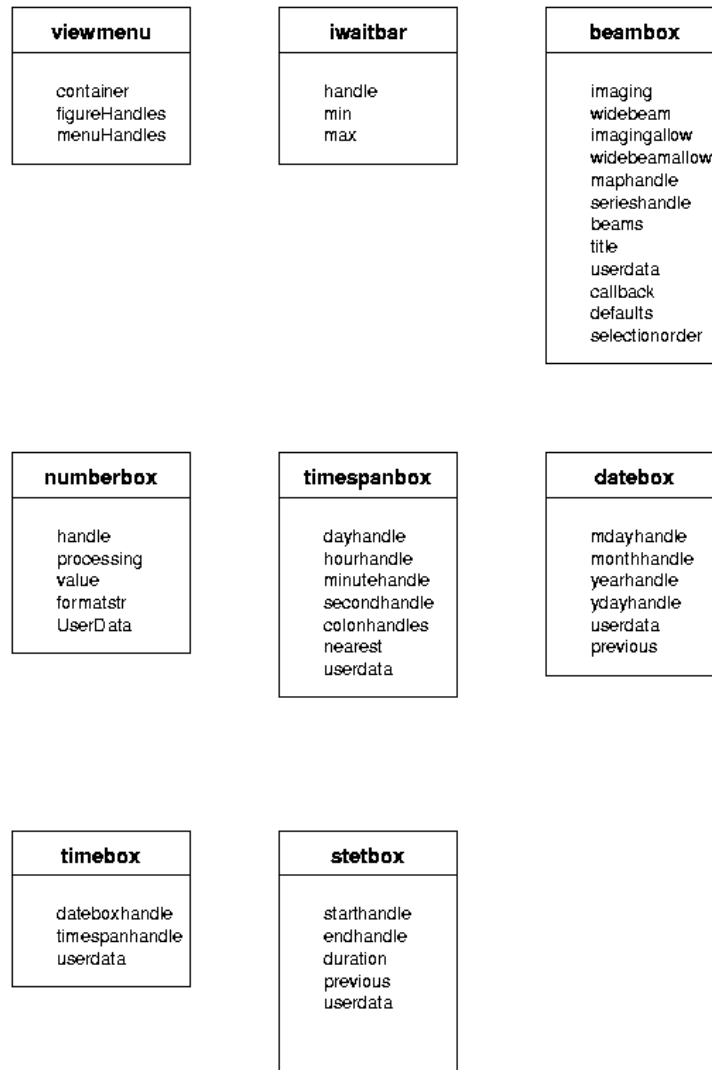
irislocation
name country geolat geolong

Some basic classes have been constructed as building blocks for the system. This includes the *time*, *timespan*, *irisint16*, *irisfiletype* and *irislocation* classes.

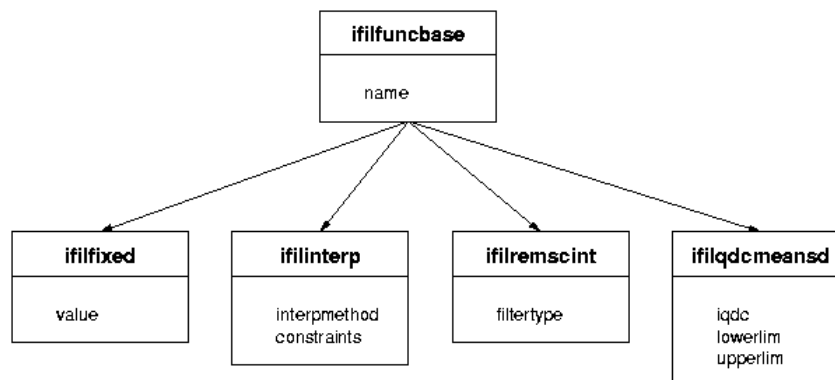
Data Classes



All the data classes are derived from *irisfilebase*. This ensures all data that can be saved to disk will contain its start and end times, resolution and location. Also included is information about the creator, the creation time and what processing has been performed on the data, to ensure the results of bad or faulty processing can be traced. Finally, the filename and path that the data was loaded with are kept for use as a default filename in case the data is saved back to disk.

GUI Classes

Several classes exist to help provide specific features for the graphical user interface.

Filter Classes

3.4 CONCLUSION

The whole project is very carefully designed; the results are accurate and they are provided very fast. It must be mentioned that the project is compatible with the IRIS-Toolkit and also future improvements are possible with no or very little modifications on the existing project design.

The Figure 3.4 presents the design of the full project.

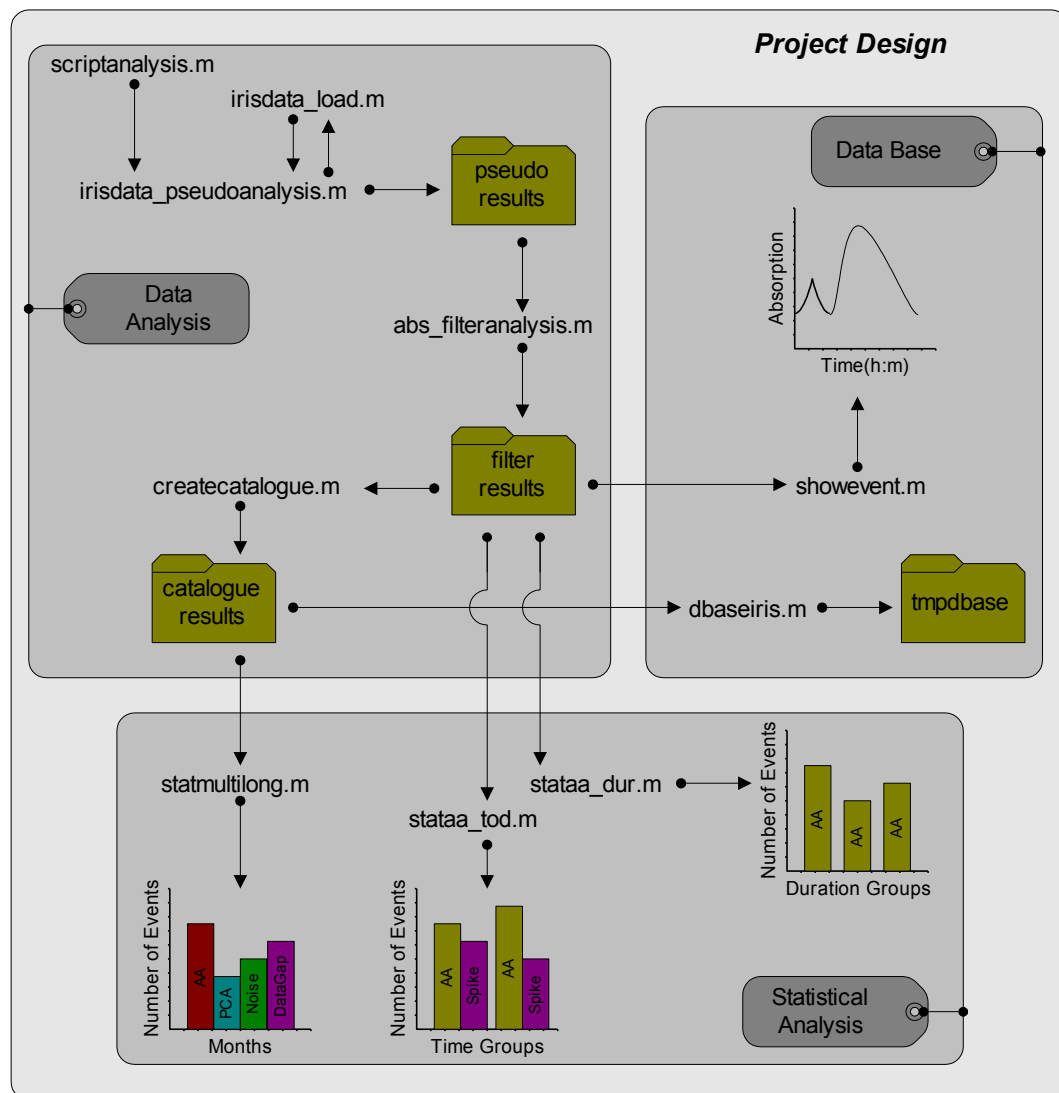


Figure 3.4: Project Design.

CHAPTER 4. PROJECT IMPLEMENTATION

4.1 INTRODUCTION

On the first half of this chapter, a detailed analysis on how to use the functions of our project is given and all the possible combinations of the users inputs and functions outputs are presented with simple examples.

The second part of this chapter, the most important algorithms that have been implemented into the project functions are presented.

The Table 4.1 lists all the functions that have been implemented in our project and the files that have been produced by the project functions.

Table 4.1: Project Functions & Files.	
DATA ANALYSIS	
<i>Functions</i>	<i>Files</i>
scriptanalysis.m	-
irisdata_load.m	-
irisdata_pseudoanalysis.m	zpseudo19941001_19941101.gn zpseudo19941101_19941201.gn ... zpseudo20000601_20000701.gn
abs_filteranalysis.m	zfilter19941001_19941101.gn zfilter19941101_19941201.gn ... zfilter20000601_20000701.gn
createcatalogue.m	zcatalogue1994.gn ... zcatalogue2000.gn
DATA BASE	
<i>Functions</i>	<i>Files</i>
dbaseiris.m	tmpdbase.gn myfilename
showevent.m	pictures (plots & images)

STATISTICAL ANALYSIS	
<i>Functions</i>	<i>Files</i>
stataa_tod.m	StatisticsI-199410.jpg StatisticsI-199411.jpg ... StatisticsI-200006.jpg
stataa_dur.m	StatisticsII-199410.jpg StatisticsII-199411.jpg ... StatisticsII-200006.jpg
statmultilong.m	StatisticsIII-1994.jpg StatisticsIII-1995.jpg ... StatisticsIII-2000.jpg ----- StatisticsIIIb-1994.jpg StatisticsIIIb-1995.jpg ... StatisticsIIIb-2000.jpg ----- StatisticsIIIc_aa-1994.jpg StatisticsIIIc_aa-1995.jpg ... StatisticsIIIc_aa-2000.jpg ----- StatisticsIIIc_pca-1994.jpg StatisticsIIIc_pca-1995.jpg ... StatisticsIIIc_pca-2000.jpg ----- StatisticsIIIc_dgap-1994.jpg StatisticsIIIc_dgap-1995.jpg ...

	StatisticsIIIc_dgap-2000.jpg

	StatisticsIIIc_noise-1994.jpg
	StatisticsIIIc_noise-1995.jpg
	...
	StatisticsIIIc_noise-2000.jpg

	StatisticsIIIc_spike-1994.jpg
	StatisticsIIIc_spike-1995.jpg
	...
	StatisticsIIIc_spike-2000.jpg

4.2 FUNCTIONS GUIDE

4.2.1 DATA ANALYSIS

As it has been discussed in the Project Design chapter, the results of the data analysis are completed in two phases. Initially the pseudo results are produced and saved in the *zpseudoStartYearMonthDay_EndYearMonthDay.gn* files and then by filtering these files we obtain the final data analysis results, which are saved in the *zfilterStartYearMonthDay_EndYearMonthDay.gn* files.

irisdata_pseudoanalysis.m

The function *irisdata_pseudoanalysis.m* accepts inputs from the user and depending on his/hers request, generates the relative output file *zpseudoStartYearMonthDay_EndYearMonthDay.gn*.

The function accepts the following inputs and its format is given as:
irisdata_pseudoanalysis(data_starttime, data_endtime, data_duration, data_resolution, beams_requested).

The *data_starttime* is a time object and represents the start time of the data analysis that the user wishes to apply on the iris data. The *data_endtime* is a time object and represents the end time of the data analysis that the user wishes to apply on the iris data. The *data_duration* is a timespan object and represents the period of time

the function will load the iris data (for fastest loading time, recommended value is 1 hour). The *data_resolution* is a timespan object and represents in what resolution the user is interested to load the iris data (for reliable and fast results, recommended value is 1 minute; if more reliable results are needed, recommended value is 1 second, but the loading time will be increased). Finally, the *beams_requested* can be a number or an array of numbers and represents the iris data of a specific beam or beams that the user wishes to load (recommended value is 50 for the widebeam or [1:50] for all the beams).

The function returns the pseudo data analysis results in the *zpseudoStartYearMonthDay_EndYearMonthDay.gn* file in ASCII format. Every time an event is found into the iris data, then a line of useful information is added into the specific file; the information are separated by a space (i.e. with a result, at the end of the data analysis the file will consists of n rows where n are the number of the events that have been found and 20 columns where each column provides a specific information for all the events). The Table 4.2 explains the information that is kept by each column into the file.

Table 4.2: Orientation of Information into the Pseudo Results File.

Columns	INFORMATION DESCRIPTION	VALID NUMBERS
01	Type of Event	1-Absorption 2-Datagap 3-Noise
02	Beam is currently analysed	1 to 50 (50 for the widebeam)
03	Start Time (Year)	1994 to 2000+
04	Start Time (Month)	1 to 12 (10 to 12 for the Year 1994)
05	Start Time (Day)	0 to 30 or 31 (depends on the Month)
06	Start Time (Hour)	0 to 24
07	Start Time (Minute)	0 to 59
08	Start Time (Second)	0 to 59
09	End Time (Year)	1994 to 2000+
10	End Time (Month)	1 to 12 (10 to 12 for the Year 1994)
11	End Time (Day)	0 to 30 or 31 (depends on the Month)
12	End Time (Hour)	0 to 24
13	End Time (Minute)	0 to 59
14	End Time (Second)	0 to 59

15	Duration (Days)	Duration of Days of the specific event
16	Duration (Hours)	Duration of Hours of the specific event
17	Duration (Minutes)	Duration of Minutes of the specific event
18	Duration (Seconds)	Duration of Seconds of the specific event
19	Max Value	Positive value for Absorption NaN for Data Gap Negative value for Noise
20	Mean Value	Positive value for Absorption NaN for Data Gap Negative value for Noise

Note that if the function will try to load an object of iris data that it does not exists into the iris server, then the system will not crash but it will write into the results file one line with 20 NaNs (1 NaN for every column) and next it will try to load the next immediately requested object of iris data. Also, in order to indicate the non-existence of the specific iris data object file, a message is displayed on the MATLAB Workspace.

Finally, in order to understand clearly how to use the function an example is given. Let we wish to apply data analysis on the iris data from the 12th of October 1995 at 21:34:00 to 23rd of February 1996 at 07:50:20 and loading the iris data in time slots of 1 day in resolution of 10 seconds and for all the beams. Note, that this is only an example that shows how to use the specific function and that in our project we were running the function for all the six years for every single month (e.g. from the 1st of October 1995 at 00:00:00 to 1st of November 1995 at 00:00:00) and loading the iris data in time slots of 1 hour in resolution of 1 minute and for the widebeam only. Continuing, the user should write the function into the MATLAB Workspace as follows: `irisdata_pseudoanalysis(time([1995 10 12 21 34 00]), time([1996 02 23 07 50 20]), timespan(1, 'd'), timespan(10, 'm'), [1:50])`. Then, the computer will start to load the requested iris data from the iris server and in parallel will analyse it and finally, the results will be stored into the *zpseudo19951012_19960223.gn* file. A sample of the contents of the file is given on Table 4.3.

Table 4.3: Contents of the *zpseudo19951012_19960223.gn* file.

01 50 1995 10 12 23 10 00 1995 10 13 01 15 30 00 02 05 30 1.34 0.95
02 50 1995 10 22 08 00 20 1995 10 22 08 19 30 00 00 19 10 NaN NaN
...
01 50 1996 02 20 04 00 50 1996 02 20 07 11 10 00 03 11 00 0.70 0.65

Special features of the *irisdata_pseudoanalysis.m* function:

- If the user will not specify on which beam or beams he/she wishes to apply the data analysis, then the function automatically sets a default value for the beams. The default value for the beams is 50 (i.e. the widebeam).
- If the user will not specify the resolution, then the function automatically sets a default value for the resolution. The default value for the resolution is 1 minute.
- If the user will not specify the duration (i.e. the period) for the analysis of the loaded iris data, then the function automatically sets a default value for the duration. The default value for the duration is 1 hour.
- If the user will not specify the start time or the end time for the data analysis, then the function returns an error message on the MATLAB Workspace. The error message is: “*Inefficient number of inputs!*”.

abs_filteranalysis.m

The function *abs_filteranalysis.m* accepts inputs from the user and depending on his/hers request, generates the relative output file *zfilterStartYearMonthDay_EndYearMonthDay.gn*.

The function accepts the following inputs and its format is given as: *abs_filteranalysis('zpseudoStartYearMonthDay_EndYearMonthDay.gn')*.

The ‘*zpseudoStartYearMonthDay_EndYearMonthDay.gn*’ is a string of characters and represents the name of any of the existing pseudo results file.

The function returns the filtered (correct) data analysis results in the *zfilterStartYearMonthDay_EndYearMonthDay.gn* file in ASCII format. If a correct event is found into the pseudo results file, then that event with its information is added into the filter results file. Also, if two or more simultaneous incorrect events are found

into the pseudo results file, then it combines those simultaneous incorrect events into a one single (correct) event and the new event with its new information is added into the filter results file. The events' information is separated by a space (i.e. with a result, at the end of the filtered data analysis the file will consists of m rows where m are the number of the (the correct plus the filtered) events that have been found and 20 columns where each column provides a specific information for all the events). The Table 4.4 explains the information that is kept by each column into the file.

Table 4.4: Orientation of Information into the Filter Results File.

Columns	INFORMATION DESCRIPTION	VALID NUMBERS
01	Type of Event	1 for General Absorption 2 for Datagap 3 for Noise 4 for Auroral Absorption + Spike Event 5 for Polar Cap Absorption
02	Beam is currently analysed	1 to 50 (50 for the widebeam)
03	Start Time (Year)	1994 to 2000+
04	Start Time (Month)	1 to 12 (10 to 11 for the Year 1994)
05	Start Time (Day)	0 to 30 or 31 (depends on the Month)
06	Start Time (Hour)	0 to 24
07	Start Time (Minute)	0 to 59
08	Start Time (Second)	0 to 59
09	End Time (Year)	1994 to 2000+
10	End Time (Month)	1 to 12 (10 to 11 for the Year 1994)
11	End Time (Day)	0 to 30 or 31 (depends on the Month)
12	End Time (Hour)	0 to 24
13	End Time (Minute)	0 to 59
14	End Time (Second)	0 to 59
15	Duration (Days)	Duration of Days of the specific event
16	Duration (Hours)	Duration of Hours of the specific event
17	Duration (Minutes)	Duration of Minutes of the specific event
18	Duration (Seconds)	Duration of Seconds of the specific event
19	Max Value	Positive value for Absorption Events

		NaN for Data Gap Negative value for Noise
20	Mean Value	Positive value for Absorption Events NaN for Data Gap Negative value for Noise

Note that the *irisdata_pseudoanalysis.m* function is searching the iris data for general absorption events, noise and datagaps. On the other hand, the *abs_filteranalysis.m* function is searching the results of the pseudo analysis files and returns in its filter results files the general absorption, the noise, the data gaps, the spike events, the auroral absorptions and the polar cap absorptions (i.e. performs a complete data analysis on the iris data).

Finally, in order to understand clearly how to use the function an example is given. Considering the example given on the *irisdata_pseudoanalysis.m* function and let we wish to apply the filter data analysis on the *zpseudo19951012_19960223.gn* file (remember the file contains the pseudo data analysis on the iris data from the 12th of October 1995 at 21:34:00 to 23rd of February 1996 at 07:50:20 and loading the iris data in time slot of 1 day in resolution of 10 seconds and for all the beams). Continuing, the user should write the function into the MATLAB Workspace as follows: *abs_filteranalysis('zpseudo19951012_19960223.gn')*. Then, the computer will load the contents of the requested file and then will analyse it and finally, the new results will be stored into the *zfilter19951012_19960223.gn* file. Observe the difference on the contents of the *zpseudo19951012_19960223.gn* file (Table 4.5a) with the contents of the *zfilter19951012_19960223.gn* file (Table 4.5b); the first two (incorrect) events from the *zpseudo19951012_19960223.gn* file have been combined into a single (correct) new event (with new characteristics) into the *zfilter19951012_19960223.gn* file.

Table 4.5a: Contents of the <i>zpseudo19951012_19960223.gn</i> file.	
01 50 1995 10 12 23 10 00 1995 10 13 01 59 00 00 02 05 30 1.34 0.95	
01 50 1995 10 13 02 00 00 1995 10 13 03 40 20 00 01 40 20 0.84 0.75	
02 50 1995 10 22 08 00 20 1995 10 22 08 19 30 00 00 19 10 NaN NaN	
...	
01 50 1996 02 20 04 00 50 1996 02 20 07 11 10 00 03 11 00 0.70 0.65	

Table 4.5b: Contents of the *zfilter19951012_19960223.gn* file.

04	50	1995	10	12	23	10	00	1995	10	13	03	40	20	00	03	45	50	1.34	0.88
02	50	1995	10	22	08	00	20	1995	10	22	08	19	30	00	00	19	10	NaN	NaN
...																			
01	50	1996	02	20	04	00	50	1996	02	20	07	11	10	00	03	11	00	0.70	0.65

createcatalogue.m

The function *createcatalogue.m* accepts inputs from the user and depending on his/hers request, generates the relative output file (*zcatalogueYear.gn*).

The function accepts the following inputs and its format is given as: *createcatalogue(Year, Months)*.

The *Year* is a number and represents the year that the user wishes to collect all the filtered (correct) results from the filter results files. Similarly the *Months* is an array of numbers and represents the months of the filtered (correct) results that the user wishes to collect for a specific year (recommended value is [1:12] for all the months of a year).

The function returns the results in the *zcatalogueYear.gn* file in ASCII format. Depending on the user's request, it loads the necessary *zfilterStartYearMonthDay_EndYearMonthDay.gn* files and merges the contents of these files into a one single *zcatalogueYear.gn* file. The contents of the new file are the same with the files before being merged (i.e. remember the Table 4.4 and the Table 4.5b); the usefulness of this function has been stated in the Project Design chapter.

Note that if the user will call the function to create a results catalogue of a specific year and months and if one of the requested *zfilterStartYearMonthDay_EndYearMonthDay.gn* files does not exists then the function will exit and the *zcatalogueYear.gn* file will not be created.

Finally, in order to understand clearly how to use the function an example is given. Let we have available the data analysis results for all the months of the year 1995 (i.e. the files *zfilter19950101_19950201.gn*, *zfilter19950201_19950301.gn*, ... *zfilter19951201_19960101.gn*) and the user wishes to create a catalogue of the data analysis results for the months 4,6,9,10,11 and 12. The user has to type the following

command on the MATLAB's Workspace: `createcatalogue(1995, [4 6 9:12])` and the function will load the requested files and the `zcatalogue1995.gn` file will be created. The contents of the file are similar with the contents of any of the `zfilterStartYearMonthDay_EndYearMonthDay.gn` files (see Table 4.5b).

Special features of the `createcatalogue.m` function:

- If the user will not specify the months that he/she wished to create the catalogue of the data analysis results, then the function automatically sets a default value for the months. The default value for the months is [1:12] (i.e. all the months).

irisdata_load.m

The function `irisdata_load.m` accepts inputs from the user and depending on his/hers request, loads the correct iris data and then returns some useful information about the loaded iris data as well.

Note that the function has not been designed for public use and is only called by other functions. However, it is useful to learn how it works and what it does.

The function accepts the following inputs and its format is given as: `irisdata_load(data_currenttime, data_duration, data_resolution, beams_requested)`.

The `data_currenttime` is a time object and represents the start date and time the user wishes to load the iris data. The `data_duration` is a timespan object and represents the duration of the loaded iris data. The `data_resolution` is a timespan object as well and represents the resolution of the iris data. Finally the `beams_requested` is a number or an array of numbers and represents the beams of the iris system. For this function it is not obligatory to give an example, once it is similar with the `irisabs` function from the `iristool v2`.

scriptanalysis.m

The helpfulness of the `scriptanalysis.m` function can be great for any user, because it calls all the data analysis functions in a special order and by using simple loops, it completes all the data analysis stages (loading of the iris data, create the pseudo analysis results and create the filtered (correct) data analysis results) for a full year or even for a certain number of months of a year. Generally it accepts the year

and the month or the months from the user and while loading the iris data it returns in parallel all the *zpseudoStartYearMonthDay_EndYearMonthDay.gn* and *zfilterStartYearMonthDay_EndYearMonthDay.gn* files.

A characteristic example of the above function can be the following. Let a user needs to apply the data analysis for the year 1995, just type *scriptanalysis(1995, [1:12])* on the MATLAB's Workspace and once the computer will finish loading the iris data the following file will be created into the user's current directory: *zpseudo19950101_19950101.gn*, *zpseudo19950201_19950301.gn*, ... *zpseudo19951201_19960101.gn* and *zfilter19950101_19950101.gn*, *zfilter19950201_19950301.gn*, ... *zfilter19951201_19960101.gn*.

Special features of the *scriptanalysis.m* function:

- If the user will not specify the months that he/she wished to apply the data analysis results, then the function automatically sets a default value for the months. The default value for the months is [1:12] (i.e. all the months).
- Every time the function finishes the loading and the data analysis parts for a specific month of a year, it returns a message on the MATLAB's Workspace about how long it took the computer to accomplish the requested process; at the termination of the analysis, it returns the total processing time for all the loaded and analysed months of a year.

4.2.2 DATA BASE

In order the Data Base functions to be fully functional, it is necessary the Data Analysis stage has been completed and all the *zfilterStartYearMonthDay_EndYearMonthDay.gn* and *zcatalogueYear.gn* files are available in any user.

dbaseiris.m

The function *dbaseiris.m* accepts inputs from the user and depending on his/hers request, generates the following output file (*tmpdbase.gn* or *myfilename*).

The function accepts the following inputs and its format is given as: *dbaseiris(Year, Months, EventType, Filename)*.

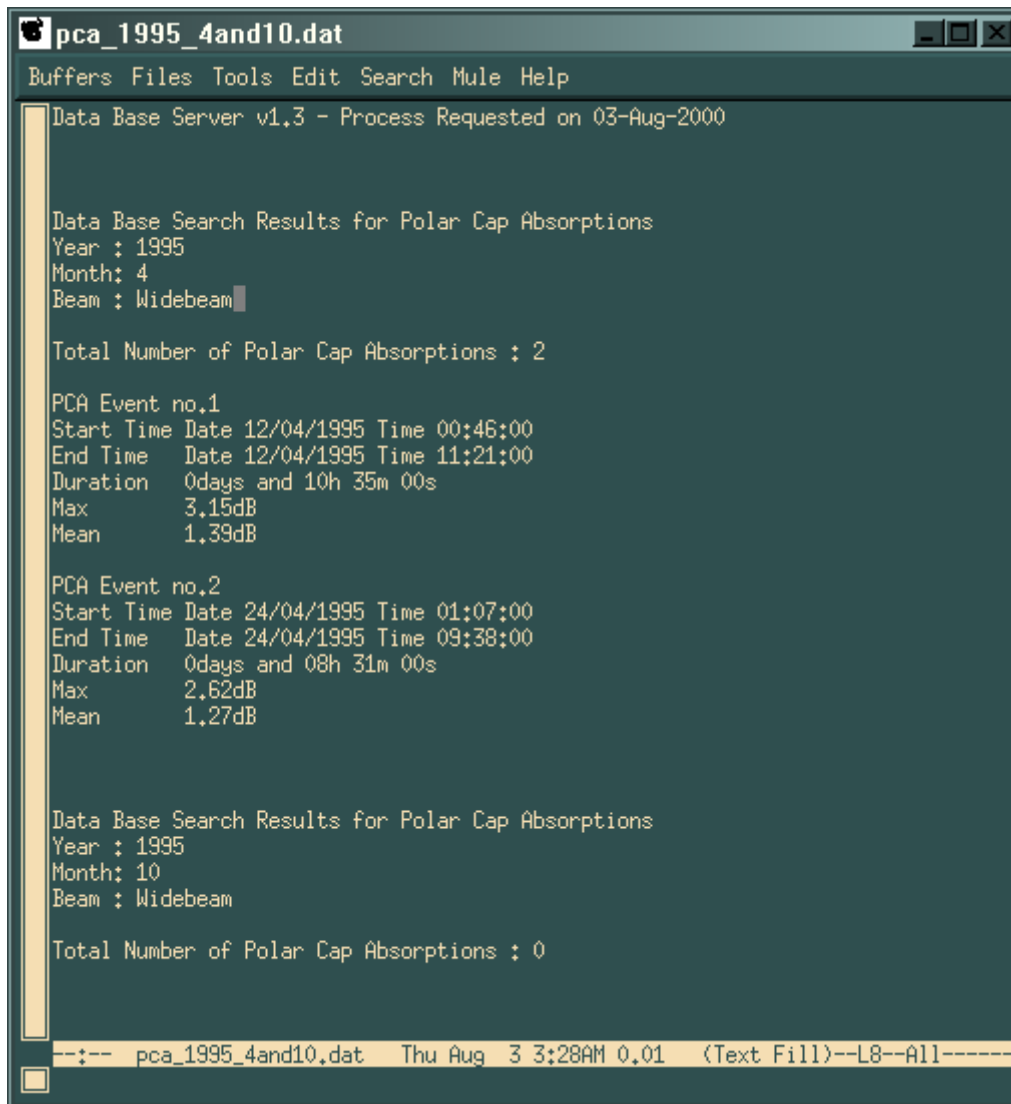
The *Year* is a number and represents the year that the user is interested to search for specific events. Similarly the *Months* is a number or an array of numbers and represents the months of the requested year that the user wishes to search for specific events. Additionally, the *EventType* is a string of characters and represents the type of event the user desires to search for; the variable *EventType* can accept the following values: ‘*spike*’ for the spike events, ‘*aa*’ for the auroral absorptions, ‘*pca*’ for the polar cap absorptions, ‘*noise*’ for the noise, ‘*datagap*’ for the data gaps and ‘*all*’ to search for all the types of events. Finally, the *Filename* is a string of characters and represents the name of the file, where the results of the database search will be saved.

The function’s search results are saved in a file. Generally, depending on the user’s request, it loads the necessary *zcatalogueYear.gn* file and applies the search for the specific year and months and looks for the requested events. In order the user to understand easily the results of the database search, the contents of the file are displayed in a user-friendly format. Initially the type of the search analysis is displayed, on the next two lines the year and the first month of the search analysis is given and a summary of the total number of events that have been occurred during that period of time is presented. Once finished with the summary, the main part of the database search analysis follows; here, every single event is displayed with some very useful information (the type of event, its start and end times, its duration and its max and mean values). After finishing with all the types of the requested events of the specific month of a year, the file continues with the database search results of the next requested month of that year (the file continues displaying the database search results in the same orientation as it has done with the first month of that year).

Note that if the user will call the function to search for a month that the data analysis results are not available, then the function will return a message on the MATLAB’s Workspace “*There are NO results available for the Month x (Year y)*” and it will continue the database search analysis for the immediately next requested month.

Finally, in order to understand clearly how to use the function an example is given. Let the user needs to learn in detail the polar cap absorptions that have been occurred in the year 1995 for the months 4 and 10 and he/she wishes the results of the database search analysis to be saved into the *pca_1995_4and10.dat* file. The function must be called as follows: *dbaseiris(1995, [4 10], ‘pca’, ‘pca_1995_4and10.dat’)* and

after few seconds the `pca_1995_4and10.dat` will be ready. The contents of the file are shown on Figure 4.1.



```

pca_1995_4and10.dat
Buffers Files Tools Edit Search Mule Help

Data Base Server v1.3 - Process Requested on 03-Aug-2000

Data Base Search Results for Polar Cap Absorptions
Year : 1995
Month: 4
Beam : Widebeam

Total Number of Polar Cap Absorptions : 2

PCA Event no.1
Start Time Date 12/04/1995 Time 00:46:00
End Time   Date 12/04/1995 Time 11:21:00
Duration   0days and 10h 35m 00s
Max        3.15dB
Mean       1.39dB

PCA Event no.2
Start Time Date 24/04/1995 Time 01:07:00
End Time   Date 24/04/1995 Time 09:38:00
Duration   0days and 08h 31m 00s
Max        2.62dB
Mean       1.27dB

Data Base Search Results for Polar Cap Absorptions
Year : 1995
Month: 10
Beam : Widebeam

Total Number of Polar Cap Absorptions : 0

--:-- pca_1995_4and10.dat Thu Aug 3 3:28AM 0.01 <Text Fill>--L8--All-----

```

Figure 4.1: Contents of the `pca_1995_4and10.dat` file.

Special features of the *dbaseiris.m* function:

- If the user will not enter the name of the file, where the results of the database search analysis to be saved, then the function saves the results in a default file. The name of the default file is *tmpdbase.gn*.
- If the user will not specify the type event that he/she wants to apply the database search analysis, then the function automatically searches for all the type of events (i.e. for the auroral absorptions, the polar cap absorptions, the spike events, the noise and the data gaps).

- If the user will not enter the months of a specific year, then the function does the search analysis for all the 12 months of the year.

showevent.m

The function **showevent.m** accepts inputs from the user and depending on his/hers request, plots the requested event in the x-y axes; also it can plot the event as a sequence of images and each image is the event as it has been seen by the IRIS system at a certain time.

The function accepts the following inputs and its format is given as: *showevent(Year, Month, Day, Hour, Minute, PlotType)*.

The inputs *Year, Month, Day, Hour* and *Minute* are numbers and represent the date and time that a specific event has occurred. The *PlotType* is a string of characters and represents the type of plot the user wishes to display the requested event; the variable *PlotType* can accept the following values: ‘*simple*’ to plot the event only on the x-y axes and ‘*advanced*’ to plot the event on the x-y axes and in a sequence of images as well.

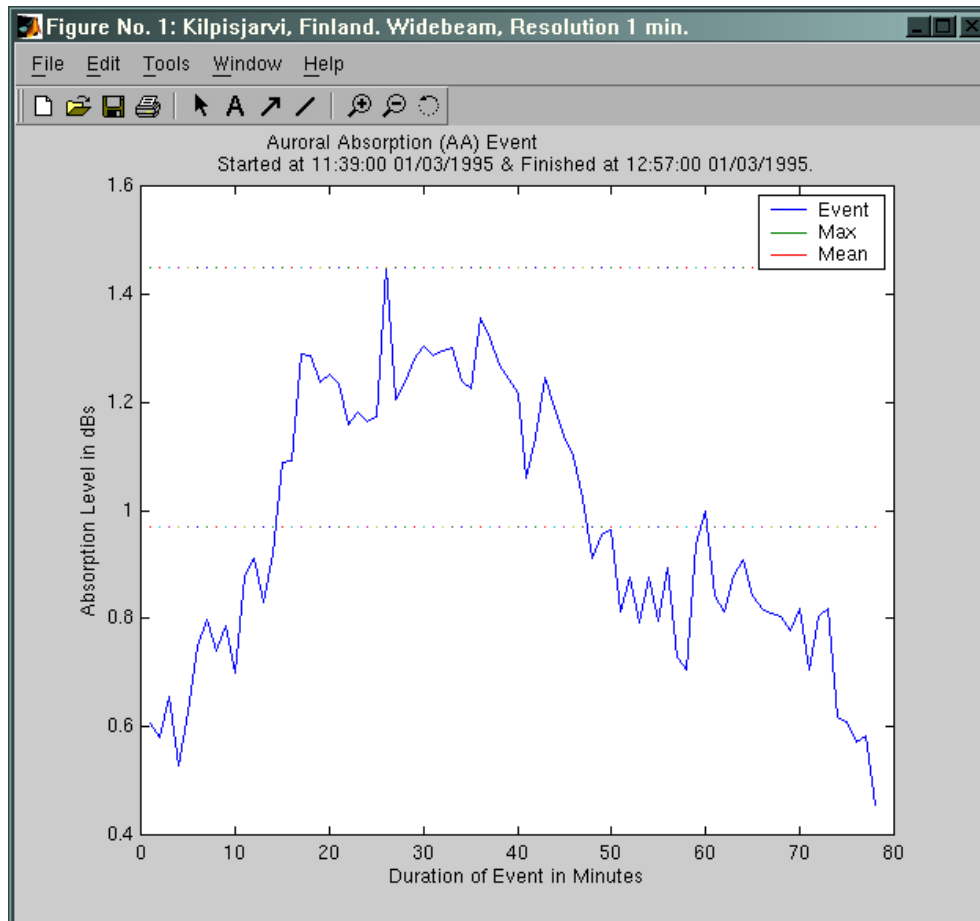
The function, depending on the user’s request, loads the necessary *zfilterStartYearMonthDay_EndYearMonthDay.gn* file and searches for the requested event and once finds it, it loads the iris data (with resolution of 1 minute) from the start and end times of the requested event and then plots it; on the plot useful information for the user are displayed (i.e. the start and end time of the event, its duration (x-axes), the absorption level (y-axes) and the maximum and mean level of the specific event.

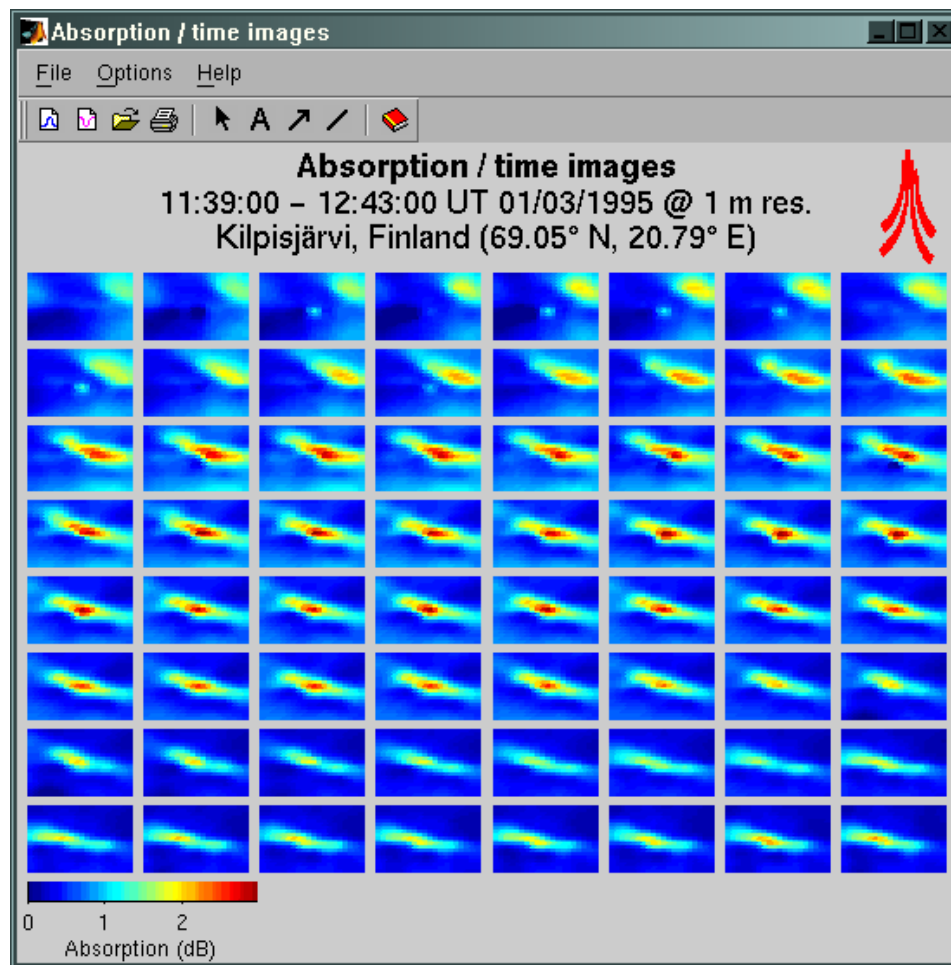
Note that if the user will call the function to search for an event that never has been occurred (i.e. the time and date that the user has been entered into the function are not the start time of an event), then the function will return a message on the MATLAB’s Workspace “*Warning! Requested Event <Day/Month/Year Hour:Minutes:Seconds> Never Occurred.*”. Therefore, it will be useful for the user to run the *dbaseiris.m* function first and once the database search results are available, then open the files where the results have been saved. Next search for the desired event and finally call the *showevent.m* function.

Finally, in order to understand clearly how to use the function an example is given. Let the user needs to plot the event that has been occurred on the 1st of March

1995 at 11:39:00, on the x-y axis and the sequence of images as well. The function must be called as follows: `showevent(1995, 03, 01, 11, 39, 'advanced')`. The two types of plots are shown on Figure 4.2a (for the Absorption vs. Duration) and Figure 4.b (for the Sequence of Images) respectively.

Figure 4.2a & 4.2b: Event occurred on the 1st of March 1995 at 11:39:00.





Note that the ‘*simple*’ mode is much faster than the ‘*advanced*’ mode; not only because the “simple” mode plots the event on the x-y axis and it does not plot it as a sequence of images, but because the ‘*simple*’ mode loads the iris data for the wide beam only and on the other hand the ‘*advanced*’ mode needs to load the iris data for all the 50 beams and of course the process requires more time for loading the iris data.

Special features of the *showevent.m* function:

- If the user will not enter the type of plot then the function sets a default value for the missing parameter; the default type of plot is the ‘*advanced*’ mode.
- If the user will not type correctly the type of plot (e.g. types: ‘*siple*’), then the function displays a message on the MATLAB’s Workspace “*The Input siple is not valid; Valid Inputs are: simple or advanced*”.

4.2.3 STATISTICAL ANALYSIS

In order the Statistical Analysis functions to be fully functional, it is necessary the Data Analysis stage has been completed and all the *zfilterStartYearMonthDay_EndYearMonthDay.gn* and *zcatalogueYear.gn* files are available in any user.

stataa_tod.m

The function *stataa_tod.m* accepts inputs from the user and depending on his/hers request, plots one or more histograms; the histogram displays the number of auroral absorptions and the number of spike events that have occurred during a specific month of a year depending on the time of their occurrence. Moreover, it prints a detailed message on the MATLAB Workspace with the exact number of the auroral absorptions and the spike events that have been occurred this period of time (i.e. it explains at the user the current histogram).

The function accepts the following inputs and its format is given as: *stataa_tod(Year, Months, Beams)*.

The inputs *Year*, *Months* are numbers and represent the year and month (or the months) that the user is interested to plot the histogram or the histograms (one histogram for each month of a year; therefore if the user request to apply the statistical analysis for the months 4 and 10 of the year 1999, then the function will plot one histogram for the month 4 year 1995 and one histogram for the month 10 year 1995). The *Beams* is a string of characters and represents on which beams the user is interested to apply the current statistical analysis; the variable *Beams* can accept the following values: ‘*widebeam*’ to count the number of events that have been occurred on the wide beam only and ‘*allbeams*’ to count the number of events that have been occurred on all the fifty beams.

The function *stataa_tod.m*, depending on the user’s request, loads the necessary *zfilterStartYearMonthDay_EndYearMonthDay.gn* file or files and searches for all the auroral absorptions and for all the spike events and orients those events depending on the time of their occurrence and then plots the results on a histogram. The events are divided into 6 categories. On the 1st category are the events that have been occurred between the hours 00:00:00 and 04:00:00; On the 2nd category are the events that have been occurred between the hours 04:00:00 and 08:00:00; On the 3rd

category are the events that have been occurred between the hours 08:00:00 and 12:00:00; On the 4th category are the events that have been occurred between the hours 12:00:00 and 16:00:00; On the 5th category are the events that have been occurred between the hours 16:00:00 and 20:00:00; On the 6th category are the events that have been occurred between the hours 20:00:00 and 24:00:00.

Note that if the user will call the function to search for a month that the data analysis results are not available, then the function will return a message on the MATLAB's Workspace "*There are NO results available for the Month x (Year y)*" and it will continue the statistical analysis on the next requested month.

Finally, in order to understand clearly how to use the function an example is given. Let the user needs to plot a histogram for the number of auroral absorptions and spike events that have been occurred on the March 1995 for the widebeam only. The function must be called as follows: `stataa_tod(1995, 03, 'widebeam')`. The histogram is shown on Figure 4.3 and on the Table 4.6 the analysis of the histogram is presented exactly as is given on the MATLAB's Workspace.

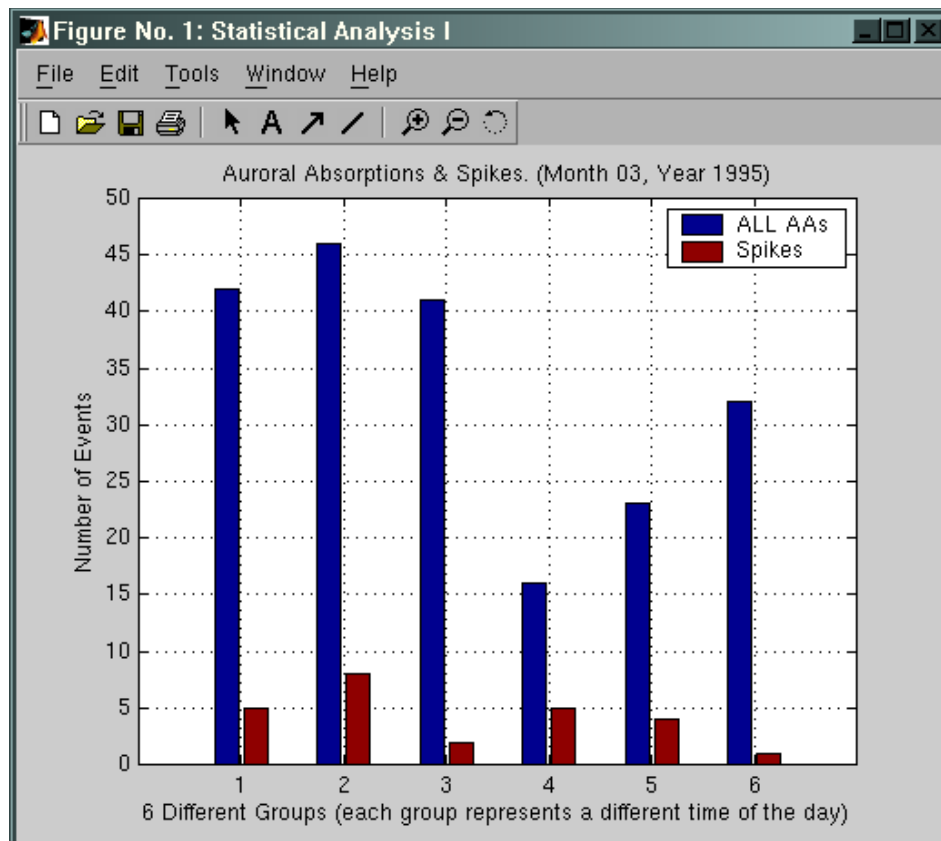


Figure 4.3: Statistical Analysis I, March 1995 (widebeam).

Table 4.6: Histogram Statistical Analysis I, March 1995 (widebeam).

Statistical Analysis

Year 1995, Month 3

<Group 1> Time of Occurrence for AA / Spike: 00:00:00 and 04:00:00

<Group 2> Time of Occurrence for AA / Spike: 04:00:00 and 08:00:00

<Group 3> Time of Occurrence for AA / Spike: 08:00:00 and 12:00:00

<Group 4> Time of Occurrence for AA / Spike: 12:00:00 and 16:00:00

<Group 5> Time of Occurrence for AA / Spike: 16:00:00 and 20:00:00

<Group 6> Time of Occurrence for AA / Spike: 20:00:00 and 24:00:00

<Group 1> Number of all AAs / Spikes: 42.0 / 5.0

<Group 2> Number of all AAs / Spikes: 46.0 / 8.0

<Group 3> Number of all AAs / Spikes: 41.0 / 2.0

<Group 4> Number of all AAs / Spikes: 16.0 / 5.0

<Group 5> Number of all AAs / Spikes: 23.0 / 4.0

<Group 6> Number of all AAs / Spikes: 32.0 / 1.0

Special features of the *stataa_tod.m* function:

- If the user will not enter the beams then the function sets a default value for the missing parameter; the default value for the beams is ‘widebeam’.
- If the user will not type correctly the beams (e.g. types: ‘wide’), then the function displays a message on the MATLAB’s Workspace “*The Input wide is not valid; Valid Inputs are: widebeam or allbeams*”.
- If the user will enter a number out of the range 1 to 12 for the months or a number 1993 or less for the year, then the function displays the following message on the MATLAB’s Workspace “*Invalid Entry for the Month X or for the Year Y*”.
- If the user will not enter the *Months* that he/she is interested to apply the statistical analysis, then the function sets a default value for the Months; the default value is [1:12] (i.e. apply the statistical analysis for all the months of the requested year).

stataa_dur.m

The function *stataa_dur.m* accepts inputs from the user and depending on his/hers request, plots one or more histograms; the histogram displays the number of auroral absorptions and the number of spike events that have occurred during a specific month of a year depending on their duration. Moreover, it prints a detailed message on the MATLAB Workspace with the exact number of the auroral absorptions and the spike events that have been occurred this period of time (i.e. it explains at the user the current histogram).

The function accepts the following inputs and its format is given as: *stataa_dur(Year, Months, Beams)*.

The inputs *Year*, *Months* are numbers and represent the year and month (or the months) that the user is interested to plot the histogram or the histograms (one histogram for each month of a year; therefore if the user request to apply the statistical analysis for the months 4 and 10 of the year 1999, then the function will plot one histogram for the month 4 year 1995 and one histogram for the month 10 year 1995). The *Beams* is a string of characters and represents on which beams the user is interested to apply the current statistical analysis; the variable *Beams* can accept the following values: 'widebeam' to count the number of events that have been occurred on the wide beam only and 'allbeams' to count the number of events that have been occurred on all the fifty beams.

The function *stataa_dur.m*, depending on the user's request, loads the necessary *zfilterStartYearMonthDay_EndYearMonthDay.gn* file or files and searches for all the auroral absorptions and for all the spike events and orients those events depending on the time of their occurrence and then plots the results on a histogram. The events are divided into 6 categories. On the 1st category are the events with duration between 2 minutes and 6 minutes (i.e. those are the spike events); On the 2nd category are the events with duration between 6 minutes and 30 minutes On the 3rd category are the events with duration between 30 minutes and 1 hour On the 4th category are the events with duration between 1 hour and 2 hours On the 5th category are the events with duration between 2 hours and 3 hours On the 6th category are the events with duration between 3 hours and 5 hours.

Note that if the user will call the function to search for a month that the data analysis results are not available, then the function will return a message on the

MATLAB's Workspace "*There are NO results available for the Month x (Year y)*" and it will continue the statistical analysis on the next requested month.

Finally, in order to understand clearly how to use the function an example is given. Let the user needs to plot a histogram for the number of auroral absorptions and spike events that have been occurred on the March 1995 for the widebeam only. The function must be called as follows: `stataa_dur(1995, 03, 'widebeam')`. The histogram is shown on Figure 4.4 and on the Table 4.7 the analysis of the histogram is presented exactly as is given on the MATLAB's Workspace.

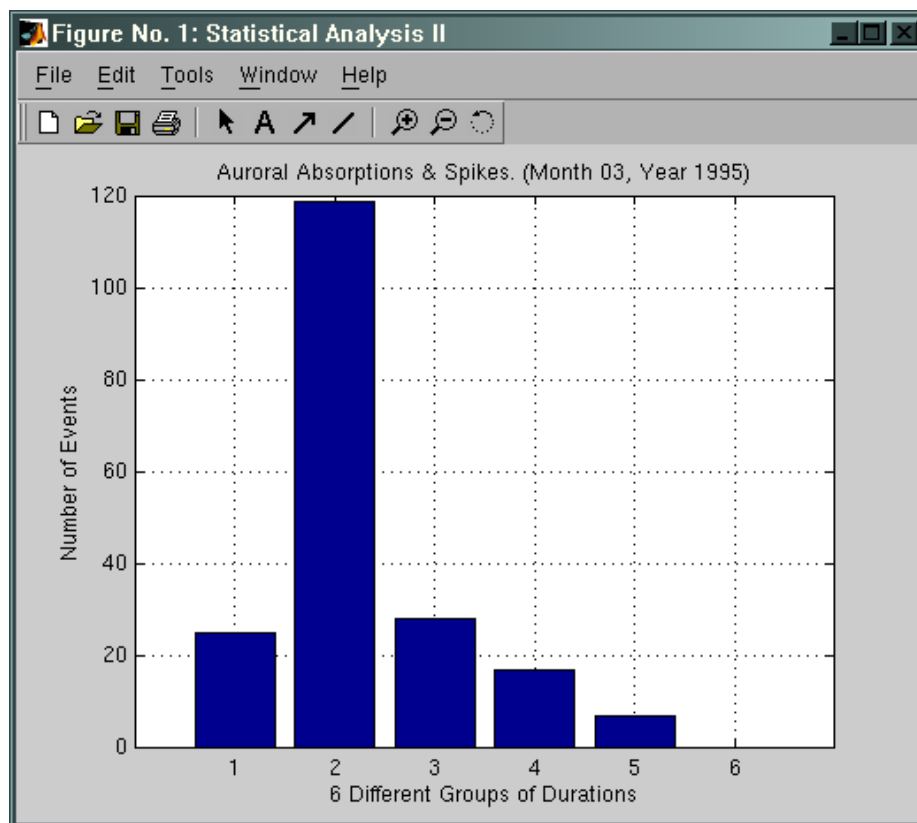


Figure 4.4: Statistical Analysis II, March 1995 (widebeam).

Table 4.7: Histogram Statistical Analysis II, March 1995 (widebeam).

Statistical Analysis

Year 1995, Month 3

<Group 1> Duration of AA (Spikes): > 02 minutes < 06 minutes

<Group 2> Duration of AA: > 06 minutes < 30 minutes

<Group 3> Duration of AA: > 30 minutes < 01 hour

<Group 4> Duration of AA: > 01 hours < 02 hours

<Group 5> Duration of AA: > 02 hours < 03 hours

<Group 6> Duration of AA: > 03 hours < 05 hours

<Group 1> Number of all AAs (Spikes): 25.0

<Group 2> Number of all AAs: 119.0

<Group 3> Number of all AAs: 28.0

<Group 4> Number of all AAs: 17.0

<Group 5> Number of all AAs: 7.0

<Group 6> Number of all AAs: 0.0

Special features of the *stataa_dur.m* function:

- If the user will not enter the beams then the function sets a default value for the missing parameter; the default value for the beams is ‘*widebeam*’.
- If the user will not type correctly the beams (e.g. types: ‘*wide*’), then the function displays a message on the MATLAB’s Workspace “*The Input wide is not valid; Valid Inputs are: widebeam or allbeams*”.
- If the user will enter a number out of the range 1 to 12 for the months or a number 1993 or less for the year, then the function displays the following message on the MATLAB’s Workspace “*Invalid Entry for the Month X or for the Year Y*”.
- If the user will not enter the *Months* that he/she is interested to apply the statistical analysis, then the function sets a default value for the Months; the default value is [1:12] (i.e. apply the statistical analysis for all the months of the requested year).

statmultilong.m

The function *statmultilong.m* accepts inputs from the user and depending on his/hers request, plots a series of histograms. The first histogram displays the number of auroral absorptions, the number of the spike events, the number of polar cap absorptions, the number of noise events and the number of data gaps that have occurred during a specific month of a year, for all the requested months. The second histogram displays exactly the same information as the first histogram does, but in a different form; it plots the results (i.e. the number of the events) of each month of a year on a sub-plot histogram. Moreover, it prints a detailed message on the MATLAB

Workspace with the exact number of all the events that have been occurred during these months of a year (i.e. it explains at the user the histograms).

The function accepts the following inputs and its format is given as: *statmultilong(Year, Months, StatisticalAnalysis, Beams)*.

The inputs *Year* and *Months* are numbers and represent the year and the months that the user is interested to plot the histograms. The *StatisticalAnalysis* is a string of characters and represents the type of the statistical analysis the user is interested to apply on the function; the *StatisticalAnalysis* can accept the following values: ‘aa’ to count the number of auroral absorptions only, ‘spike’ to count the number of the spike events only, ‘pca’ to count the number of the polar cap absorptions only, ‘noise’ to count the number of the noise events only, ‘datagap’ to count the number of the data gaps only and ‘all’ to count the number of all the events in parallel. The *Beams* is a string of characters and represents on which beams the user is interested to apply the current statistical analysis; the variable *Beams* can accept the following values: ‘widebeam’ to count the number of events that have been occurred on the wide beam only and ‘allbeams’ to count the number of events that have been occurred on all the fifty beams.

The function *statmultilong.m*, depending on the user’s request, loads the necessary *zcatalogueYear.gn* file and searches for the requested type of event (or events) depending on the requested months of a year of their occurrence and then plots the results on a histogram.

Note that if the user will call the function to search for a month that the data analysis results are not available, then the function will return a message on the MATLAB’s Workspace “*There are NO results available for the Month x (Year y)*” and it will continue the statistical analysis on the next requested month.

Finally, in order to understand clearly how to use the function an example is given. Let the user needs to plot a histogram for the number of all the types of events that have been occurred on all the twelve months of the year 1995 for the wide beam only. The function must be called as follows: *statmultilong(1995, [1:12], ‘all’, ‘widebeam’)*. The histograms are shown on the Figures 4.5 & 4.6 and on the Table 4.8 the analysis of the histogram are presented exactly as is given on the MATLAB’s Workspace.

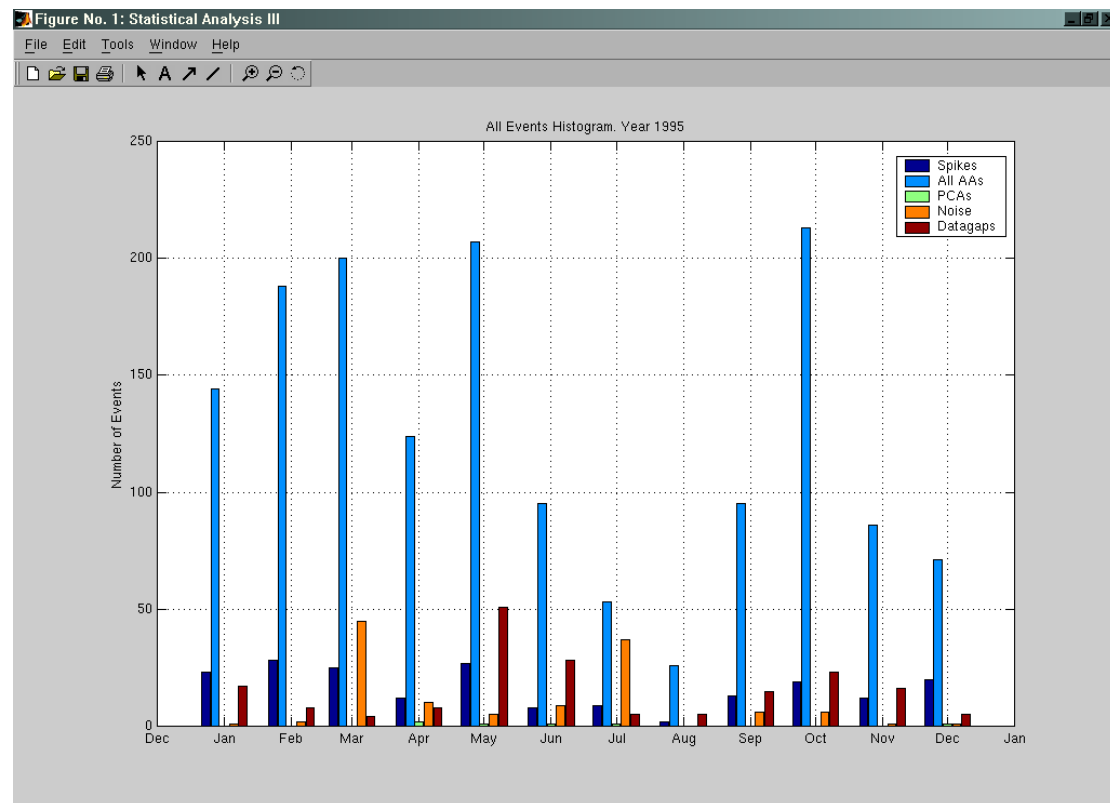


Figure 4.5: Statistical Analysis III, All the Events, Year 1995 (widebeam).

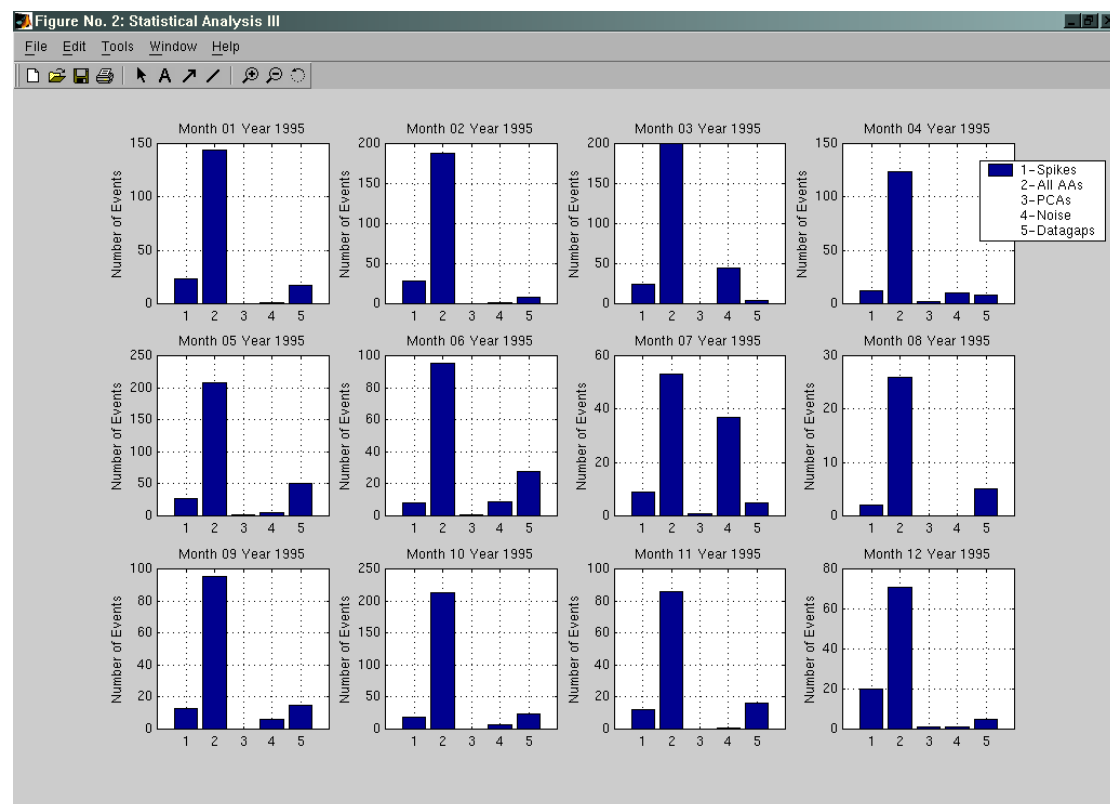


Figure 4.6: Statistical Analysis III, All the Events, Year 1995 (widebeam).

Table 4.8: Histogram Statistical Analysis III, All the Events, Year 1995 (widebeam).

Statistical Analysis III, Year 1995

Month 1

Total Number of Spikes : 23.0

Total Number of AAs : 144.0

Total Number of PCAs : 0.0

Total Number of Noise : 1.0

Total Number of Datagaps : 17.0

...

Month 5

...

Month12

Total Number of Spikes : 20.0

Total Number of AAs : 71.0

Total Number of PCAs : 1.0

Total Number of Noise : 1.0

Total Number of Datagaps : 5.0

Special features of the *statmultilong.m* function:

- If the user will not enter the beams then the function sets a default value for the missing parameter; the default value for the beams is ‘*widebeam*’.
- If the user will not type correctly the beams (e.g. types: ‘*wide*’), then the function displays a message on the MATLAB’s Workspace “*The Input wide is not valid; Valid Inputs are: widebeam or allbeams*”.
- If the user will not enter the type of the statistical analysis then the function sets a default value for the missing parameter; the default value for the type of the statistical analysis is ‘*all*’.
- If the user will not type correctly the type of the statistical analysis (e.g. types ‘*auroral*’), then the function displays a message on the MATLAB’s

Workspace “*The Input auroral is not valid; Valid Inputs are: aa, spike, pca, noise, datagap and all*”.

- If the user will enter a number out of the range 1 to 12 for the months or a number 1993 or less for the year, then the function displays the following message on the MATLAB’s Workspace “*Invalid Entry for the Month X or for the Year Y*”.
- If the user will not enter the *Months* that he/she is interested to apply the statistical analysis, then the function sets a default value for the Months; the default value is [1:12] (i.e. apply the statistical analysis for all the months of the requested year).

4.3 FUNCTIONS ANALYSIS

In order for a developer or even for a user to understand the algorithms that have been implemented into a MATLAB source code (functions), he/she must go through the MATLAB source code (**Appendix A**) and then the **comments into the MATLAB code will guide** him/her and give an explanation on how the algorithms have been designed and implemented as well. Below some (and not all) major algorithms are explained.

4.3.1 EVENTS’ RULES

All the code for the general absorption, noise and data gap have been designed and implemented into the *irisdata_pseudoanalysis.m* function (**Appendix A**), but the different types of absorptions, noise and data gap events (i.e. the auroral absorptions, the spike events, the polar cap absorptions, the noise and the data gaps) have been designed and implemented into the *abs_filteranalysis.m* function (**Appendix A**).

Auroral Absorption

The characteristics of the auroral absorption are presented on Table 4.9, the pseudo code for this event is given in Table 4.10; note that the pseudo code for the auroral absorption is only directional for the programmer; for more detail of the

algorithms code, see the *irisdata_pseudoanalysis.m* and *abs_filteranalysis.m* functions.

Table 4.9: Characteristics of the Auroral Absorptions.

Type	Min Duration	Max Duration	Min Value	Max Value
Auroral Absorption	6 minutes	5 hours	0.5 dB	- dB

Table 4.10: Pseudo Code for the Auroral Absorptions.

if
 (duration of event > 6 minutes *AND* duration of event < 5 hours)
AND (maximum event's value > 0.5)
then
 Auroral Absorption True, add some counters.

Spike Event

The characteristics of the spike event are presented on Table 4.11, the pseudo code for this event is given in Table 4.12; note that the pseudo code for the spike event is only directional for the programmer; for more detail of the algorithms code, see the *irisdata_pseudoanalysis.m* and *abs_filteranalysis.m* functions.

Table 4.11: Characteristics of the Spike Events.

Type	Min Duration	Max Duration	Min Value	Max Value
Spike Event	2 minutes	6 minutes	1 dB	- dB

Table 4.12: Pseudo Code for the Spike Events.

if
 (duration of event > 2 minutes *AND* duration of event < 6 minutes)
AND (maximum event's value > 1)
then
 Spike Event True, add some counters.

Polar Cap Absorption

The characteristics of the polar cap absorption are presented on Table 4.13, the pseudo code for this event is given in Table 4.14; note that the pseudo code for the polar cap absorption is only directional for the programmer; for more detail of the algorithms code, see the *irisdata_pseudoanalysis.m* and *abs_filteranalysis.m* functions.

Table 4.13: Characteristics of the Polar Cap Absorptions.				
Type	Min Duration	Max Duration	Min Value	Max Value
Polar Cap Absorption	8 hours	10 days	1 dB	- dB

Table 4.14: Pseudo Code for the Polar Cap Absorptions.
<i>If</i> (duration of event > 8 hours <i>AND</i> duration of event < 10 days) <i>AND</i> (maximum event's value > 1) <i>then</i> Polar Cap Absorption True, add some counters.

Noise

The characteristics of the noise are presented on Table 4.15, the pseudo code for this event is given in Table 4.16; note that the pseudo code for the noise is only directional for the programmer; for more detail of the algorithms code, see the *irisdata_pseudoanalysis.m* and *abs_filteranalysis.m* functions.

Table 4.15: Characteristics of the Noise Events.				
Type	Min Duration	Max Duration	Min Value	Max Value
Noise	1 minutes	-	- dB	-1 dB

Table 4.16: Pseudo Code for the Noise Events.

```

if
    (duration of event > 1 minutes)
    AND (minimum event's value < -1)
then
    Noise True, add some counters.

```

Data Gap

The characteristics of the Data Gap are presented on Table 4.17, the pseudo code for this event is given in Table 4.18; note that the pseudo code for the Data Gap is only directional for the programmer; for more detail of the algorithms code, see the *irisdata_pseudoanalysis.m* and *abs_filteranalysis.m* function.

Table 4.17: Characteristics of the Data Gaps.

Type	Min Duration	Max Duration	Value
Data Gaps	-	-	NaN dB

Table 4.18: Pseudo Code for the Data Gaps.

```

If (event's value is NaN)
then
    Datagap True, add some counters.

```

4.3.2 EVENTS' MEAN VALUE

As it has been mentioned earlier, the *irisdata_pseudoanalysis.m* function loads and analyses one hour of the iris data each time and saves the results in a *zpseudoStartYearMonthDay_EndYearMonthDay.gn* file; the reason of having the *abs_filteranalysis.m* function is because of the incorrect results of the *irisdata_pseudoanalysis.m* function analysis and therefore some kind of filtering was needed in order to make a correct analysis on the iris data. At this stage a problem

with the mean value of the filtered events has been encountered and it will be explained with the following example.

Let three incorrect simultaneous events have been found during the pseudo analysis with the following characteristics:

The first event started on the 20th of September 1995 at 23:20:00 and finished at 23:59:00 (i.e. with duration of 40 minutes), max value 1.7dBs and mean value 1.3dBs. The second event started on the 21st of September 1995 at 00:00:00 and finished at 23:59:00 (i.e. with duration of 23 hours and 59 minutes), max value 0.7dBs and mean value 0.6dBs. The third started on the 22nd of September 1995 at 00:00:00 and finished at 03:10:00 (i.e. with duration of 3 hours and 10 minutes), max value 1.2dBs and mean value 0.9dBs.

After the filtering, the three incorrect events are going to be combined into one single new event with the following characteristics:

The new filtered event started on the 20th of September 1995 at 23:20:00 (i.e. the start time of the first incorrect event) and finished on the 22nd of September 1995 at 03:10:00 (i.e. the end time of the last incorrect event), duration of 1 days 3 hours and 50 minutes (i.e. the sum of durations of all three incorrect events), max value 1.7dBs (i.e. the max value of the three incorrect events) and mean value 0.6095dBs.

The mean value of the new event is not so simple to found, because its value depends on the individual mean values of the three incorrect events, the duration of each incorrect events and the total duration of the three incorrect events as well. The following algorithm shows how the mean value of the new (filtered) event has been calculated.

```

» dur_tot = 1*24*60 + 3*60 + 50;
» dur_1 = 40;
» dur_2 = 1*24*60;
» dur_3 = 3*60 + 10;
» mean_1 = 1.3;
» mean_2 = 0.6;
» mean_3 = 0.9;
» mean_tot = (dur_1/dur_tot)*mean_1 + (dur_2/dur_tot)*mean_2 +
(dur_3/dur_tot)*mean_3;
» mean_tot

```



```
mean_tot =
```

```
0.6509
```

Note that the algorithm for the mean value that has been used into the *abs_filteranalysis.m* function is much more complicated because it should work for different number of simultaneous incorrect events, with different durations and different mean values.

4.3.3 OTHER ALGORITHMS

There are more complex and longer algorithms than the algorithms that have been discussed on the previous sections and it is not possible to analyse them into the report, the only solution to understand them is to follow the comments inside the MATLAB source code (**Appendix A**).

The three functions used for the statistical analysis (*stataa_tod.m*, *stataa_dur.m* and *statmultilong.m*) are very efficiently designed and depending on the users' requests behaves differently by returning relative results. The same behaviour exists with the function for the data base search (*dbaseiris.m*).

4.4 CONCLUSION

This chapter can be divided into two main parts. The first part provides useful information for any person (i.e. for any user) who is interested to use the functions efficiently and to be able to understand the outputs that are obtained by them. On the other hand, the second part provides useful information for any person (i.e. for any programmer) who wants to understand in detail the function algorithms and wishes to learn how the project functions have been implemented; therefore to be able to implement similar functions for his/hers project, further to be able to create new functions for our project or even to modify the existing functions.

CHAPTER 5. PROJECT STATISTICAL RESULTS

The complete catalogue of the Statistical Analysis Results is fully listed on **Appendix B** (tables) and on **Appendix C** (histograms).

5.1 Long Term Statistics – Type I

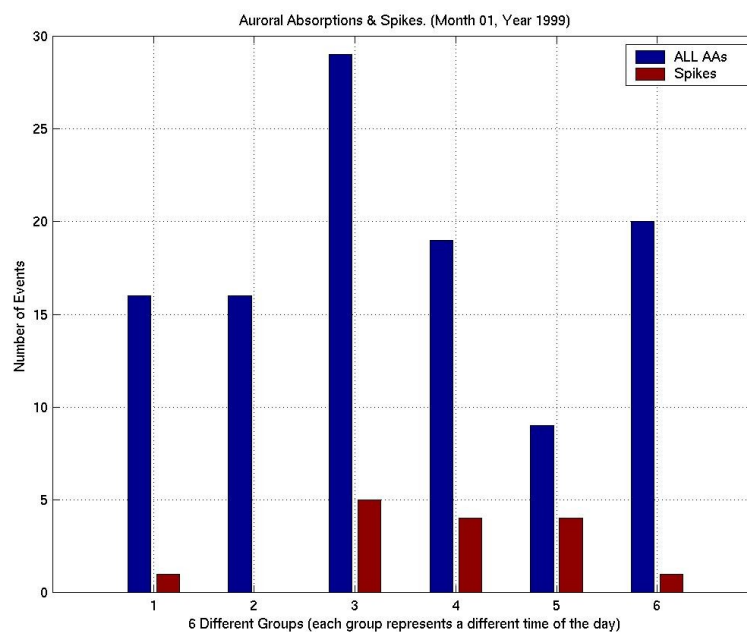
Once the Data Analysis of the iris data is finished and the results files (*zfilterStartYearMonthDay_EndYearMonthDay.gn* and *zcatalogueYear.gn*) have been created, then it is possible the first type of the Statistical Analysis to be executed by using the *stataa_tod.m* function.

The aim of this type of Statistical Analysis is to find, for every month of all the six years (1994-2000), the number of the auroral absorption events that depend on the time of their occurrence.

Next a sample of the results that have been obtained by the first statistical analysis is given. The Table 5.4.1 contains the results of the Statistical Analysis – Type I for the year 1999 and the Figure 5.4.1 displays the results of the Statistical Analysis – Type I for the month January year 1999.

Table 5.4.1: Sample of the Statistical Analysis Results – Type I, Year 1999.

Statistical Analysis I						
Year 1999	Number of Auroral Absorptions / Number of Spikes Depending on the Time of their Occurrence.					
Month	00:00:00 04:00:00	04:00:00 08:00:00	08:00:00 12:00:00	12:00:00 16:00:00	16:00:00 20:00:00	20:00:00 24:00:00
01	16 / 01	16 / 00	29 / 05	19 / 04	09 / 04	20 / 01
02	08 / 01	14 / 02	36 / 00	03 / 00	09 / 00	08 / 02
03	49 / 08	39 / 07	46 / 10	10 / 01	09 / 01	35 / 02
04	37 / 03	40 / 06	53 / 11	20 / 03	22 / 04	29 / 01
05	31 / 04	30 / 05	41 / 10	48 / 07	17 / 03	23 / 02
06	53 / 13	67 / 13	66 / 08	69 / 11	60 / 14	60 / 06
07	27 / 10	36 / 12	31 / 10	16 / 05	30 / 16	40 / 11
08	55 / 08	40 / 05	42 / 06	07 / 01	21 / 02	40 / 08
09	41 / 04	56 / 05	41 / 08	11 / 01	14 / 02	45 / 06
10	47 / 02	53 / 05	68 / 19	08 / 01	23 / 04	46 / 08
11	32 / 04	58 / 06	319 / 177	79 / 40	20 / 01	40 / 07
12	31 / 02	61 / 11	57 / 07	17 / 02	20 / 02	30 / 02

**Figure 5.4.1:** Sample of the Statistical Analysis Results – Type I, January 1999.

The results of this Statistical Analysis are presented as a series of histograms (one histogram for every single month of a year) and all these histograms are listed on **Appendix C from the Figure 5.1.1 until the Figure 5.1.69**. Additionally a detailed summary of all these histograms is given on **Appendix B from the Table 5.1.1 until the Table 5.1.7**.

5.2 Long Term Statistics – Type II

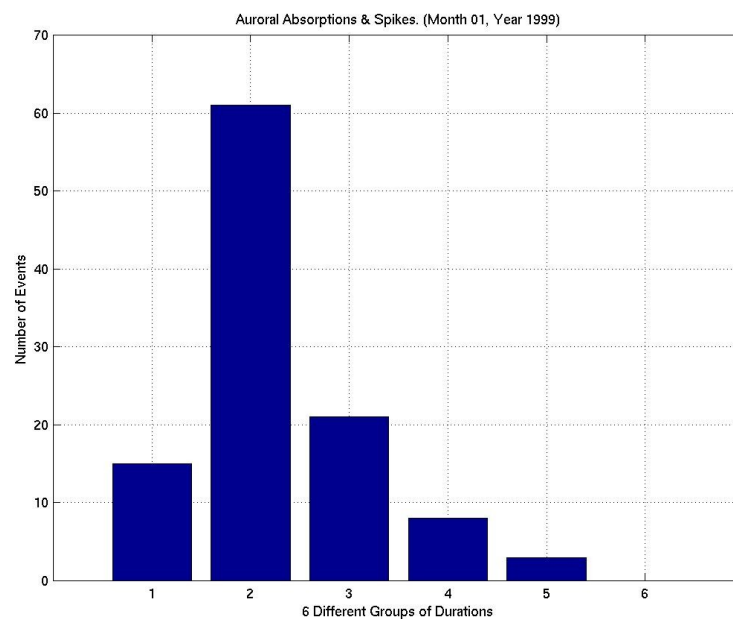
Once the Data Analysis of the iris data is finished and the results files (*zfilterStartYearMonthDay_EndYearMonthDay.gn* and *zcatalogueYear.gn*) have been created, then it is possible the second type of the Statistical Analysis to be executed by using the *stataa_dur.m* function.

The aim of this type of Statistical Analysis is to find, for every month of all the six years (1994-2000), the number of the auroral absorption events that depend on their duration.

Next a sample of the results that have been obtained by the second statistical analysis is given. The Table 5.4.2 contains the results of the Statistical Analysis – Type II for the year 1999 and the Figure 5.4.2 displays the results of the Statistical Analysis – Type II for the month January year 1999.

Table 5.4.2: Sample of the Statistical Analysis Results – Type II, Year 1999.

Statistical Analysis II						
Year 1999	Number of Auroral Absorptions Depending on their Duration.					
Month	>2mins <6mins	>6mins <30mins	>30mins <1hours	>1hours <2hours	>2hours <3hours	>3hours <5hours
01	15	61	21	08	03	00
02	05	44	14	12	02	00
03	29	93	33	23	10	00
04	28	112	28	22	07	00
05	31	123	19	12	05	00
06	65	272	19	15	03	00
07	64	99	12	04	01	00
08	30	132	24	15	02	00
09	26	124	24	23	06	00
10	39	128	42	28	05	00
11	235	246	36	22	07	00
12	26	144	21	17	03	00

**Figure 5.4.2:** Sample of the Statistical Analysis Results – Type II, January 1999.

The results of this Statistical Analysis are presented as a series of histograms (one histogram for every single month of a year) and all these histograms are listed on **Appendix C from the Figure 5.2.1 until the Figure 5.2.69**. Additionally a detailed summary of all these histograms is given on **Appendix B from the Table 5.2.1 until the Table 5.2.7**.

5.3 Long Term Statistics – Type III

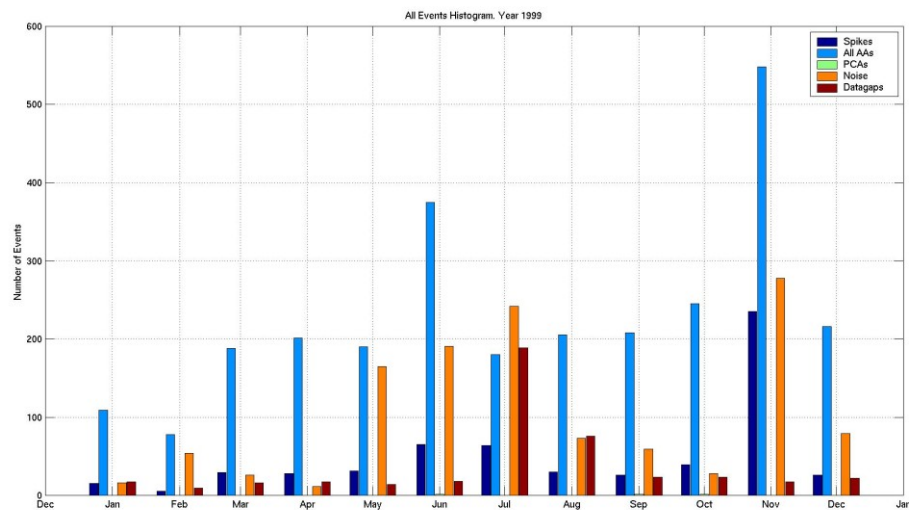
Once the Data Analysis of the iris data is finished and the results files (*zfilterStartYearMonthDay_EndYearMonthDay.gn* and *zcatalogueYear.gn*) have been created, then it is possible the third type of the Statistical Analysis to be executed by using the *statmultilong.m* function.

The aim of this type of Statistical Analysis is to find, for every month of all the six years (1994-2000), the number of the auroral absorption events, the spike events, the number of polar cap absorptions, the number of noise events and the number of data gaps as well.

Next a sample of the results that have been obtained by the third statistical analysis is given. The Table 5.4.3 contains the results of the Statistical Analysis – Type III for the year 1999 and the Figure 5.4.3 displays the results of the Statistical Analysis – Type III for the month January year 1999.

Table 5.4.3: Sample of the Statistical Analysis Results – Type III, Year 1999.

Statistical Analysis III					
Year 1999	Number of Events				
Month	Spikes	AA	PCA	Noise	DataGaps
01 - January	15	109	00	16	17
02 - February	05	78	00	54	09
03 - March	29	188	00	26	16
04 - April	28	201	00	11	17
05 - May	31	190	00	165	14
06 - June	65	375	01	191	18
07 - July	64	180	00	242	189
08 - August	30	205	00	73	76
09 - September	26	208	01	59	23
10 - October	39	245	01	28	23
11 - November	235	548	00	278	17
12 - December	26	216	00	79	22



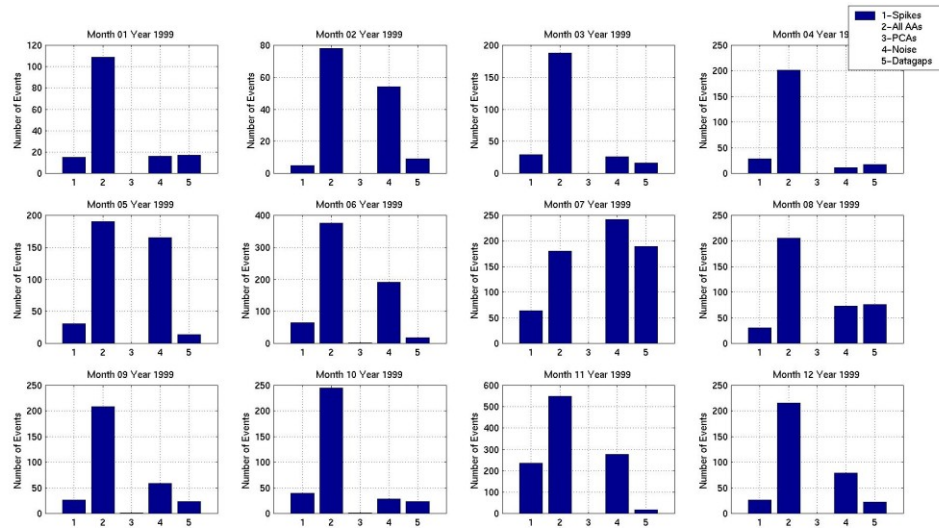


Figure 5.4.3: Sample of the Statistical Analysis Results – Type III, January 1999.

The results of this Statistical Analysis are presented as a series of histograms (one histogram for every single month of a year) and all these histograms are listed on **Appendix C** from the **Figure 5.3.1** until the **Figure 5.3.7** (for all the events), from the **Figure 5.3.1b** until the **Figure 5.3.7b** (for all the events), from the **Figure 5.3.1c-1** until the **Figure 5.3.7c-1** (for the auroral absorptions), from the **Figure 5.3.1c-2** until the **Figure 5.3.7c-2** (for the spike events), from the **Figure 5.3.1c-3** until the **Figure 5.3.7c-3** (for the polar cap absorptions), from the **Figure 5.3.1c-4** until the **Figure 5.3.7c-4** (for the noise events) and from the **Figure 5.3.1c-5** until the **Figure 5.3.7c-5** (for the data gaps). Additionally a detailed summary of all these histograms is given on **Appendix B** from the **Table 5.3.1** until the **Table 5.3.7**.

CHAPTER 6. PCAs DURATION DISTRIBUTION

Once the results of the Statistical Analysis part have been produced (**Appendix B** and **Appendix C**), then we can proceed on the analyses of those results. Note that it is possible to apply many different types of analysis on the Statistical Analysis results that have been produced by our project. In this chapter the monthly distribution of the polar cap absorptions duration will be examined and analysed.

After collecting all the useful information for the polar cap absorptions from the Statistical Analysis results, plus the information taken from the Data Base results, one graph has been produced showing the average duration of the polar cap absorptions for every month for the years 1994-2000 (Table 6.4.1) and a second graph that shows in more detail how the duration of the polar cap absorptions varies on each month for each year, the average and the total duration for all the years is displayed as well (Table 6.4.2).

Table 6.4.1: Average Duration for PCA events on each Month (Years 1994-2000).

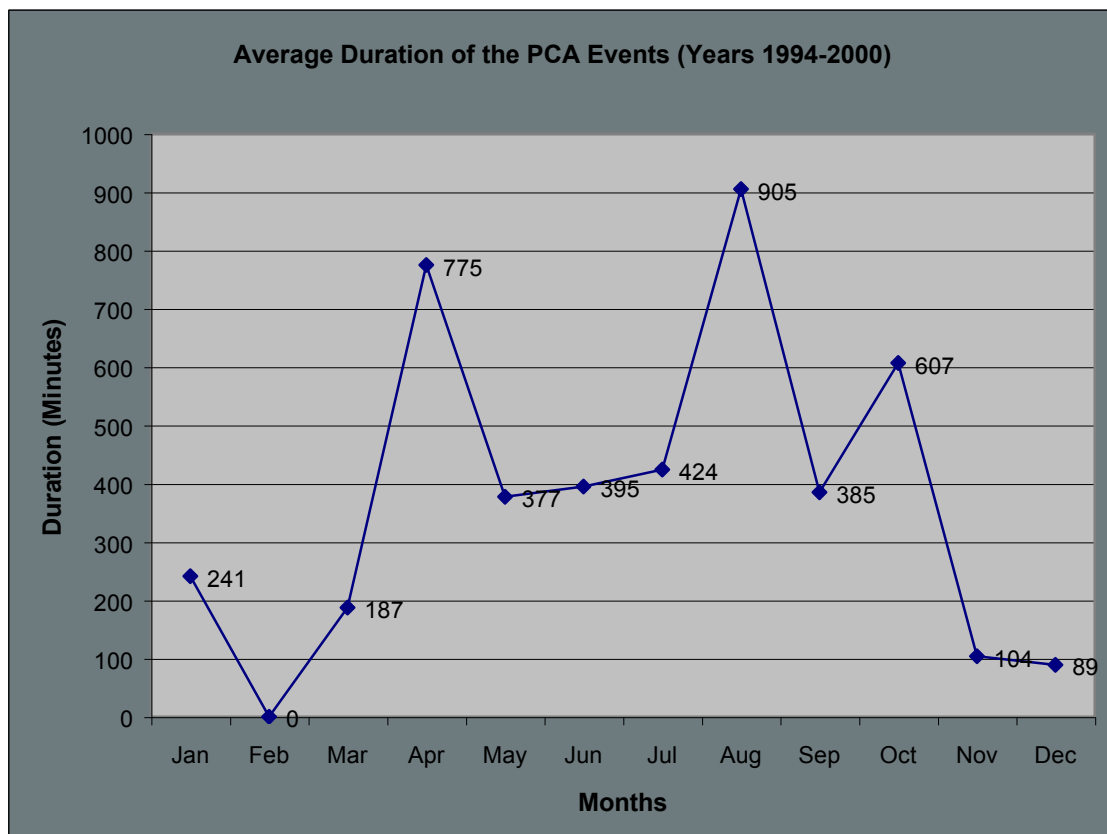
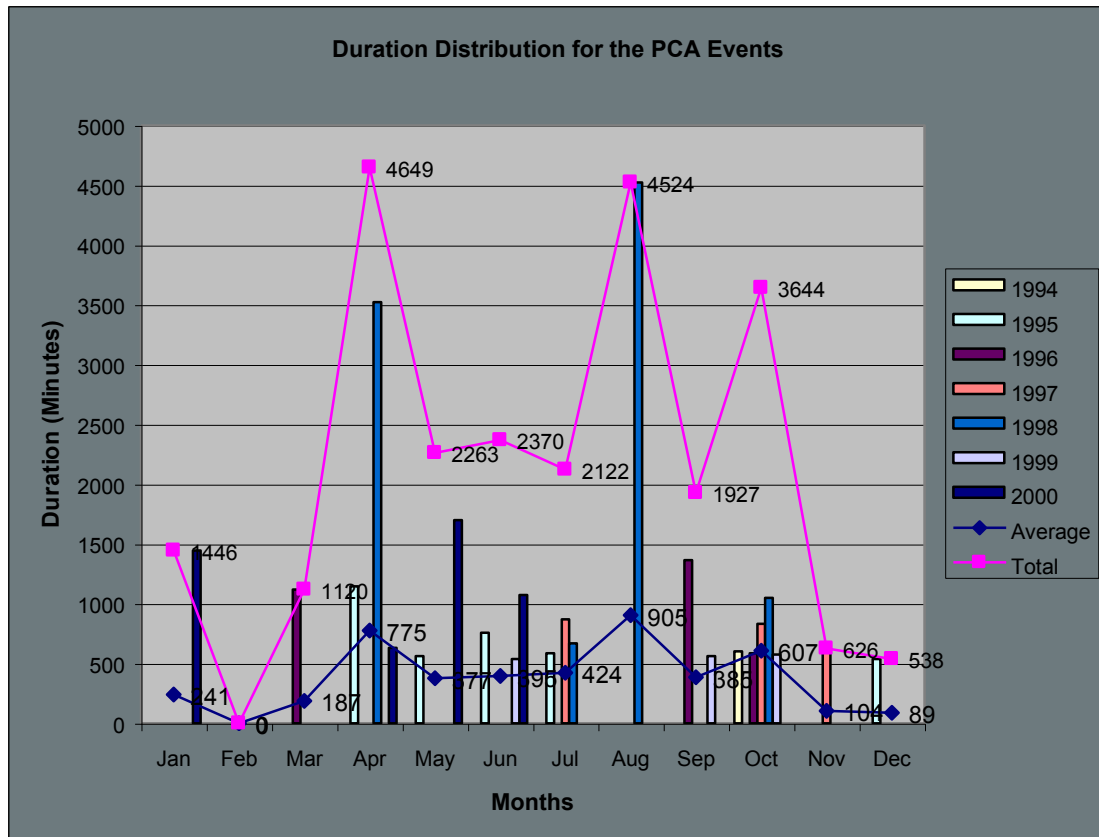


Table 6.4.2: Duration Distribution for PCA Events on each Month for every Year.

CHAPTER 7. AURORAL ABSORPTIONS DISTRIBUTION

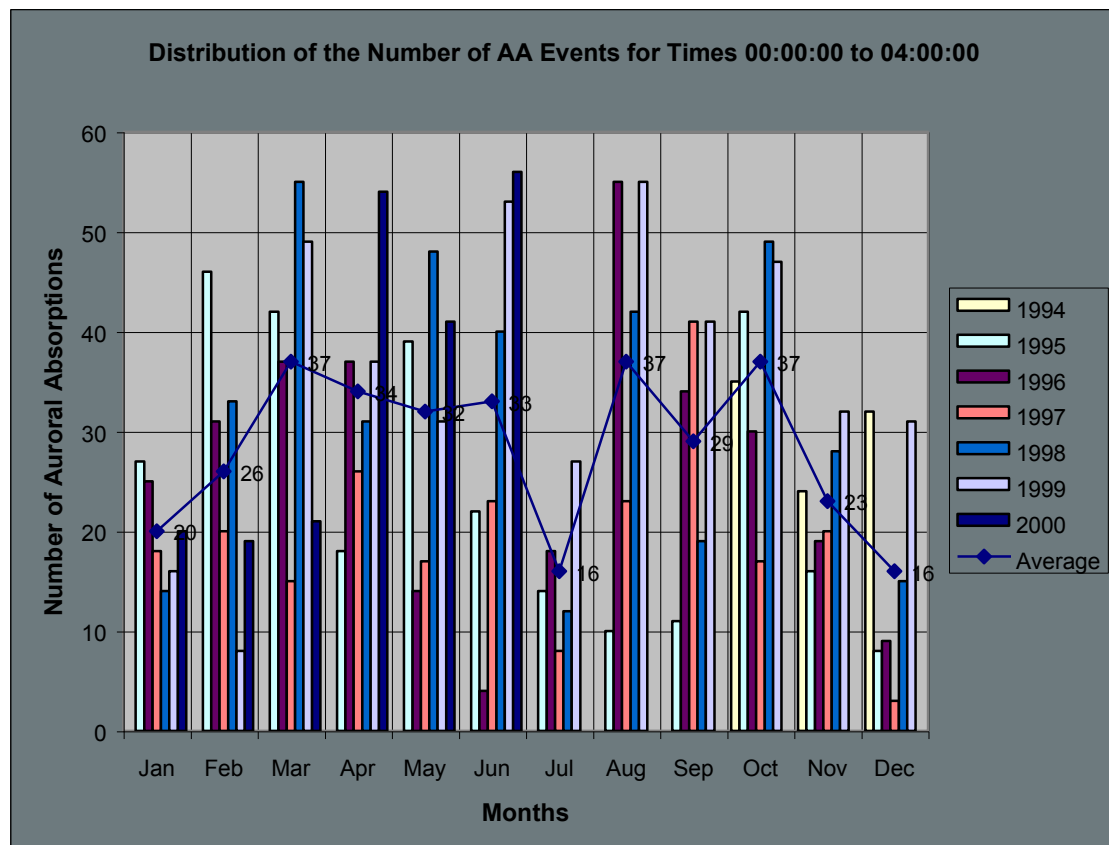
Similarly with the chapter 6, in chapter 7 a second type of analysis of the Statistical Analysis results follows. In this chapter the distribution of the number of the auroral absorptions that depend on the time of the day is examined and analysed and the distribution of the number of the auroral absorptions that depend on their duration is examined and analysed as well.

7.1 Auroral Absorption Events' Time of Day Distribution

Collecting all the useful information for the auroral absorptions from the Statistical Analysis results, a graph has been produced showing how the number of the auroral absorptions that depend on the time of their occurrence varies on each month for every year.

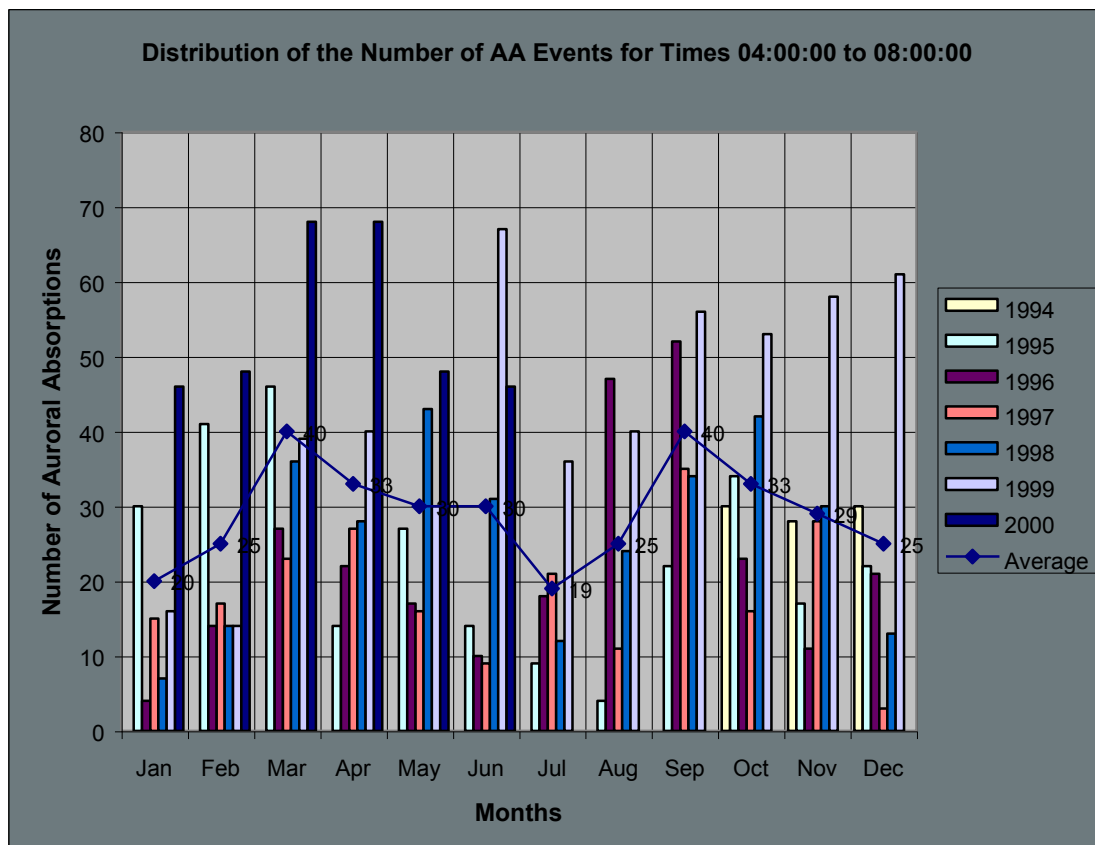
The Table 7.1.1 shows the average number of auroral absorptions that have occurred during the 00:00:00 and 04:00:00 hours on each day on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions that have occurred during the 00:00:00 and 04:00:00 hours, for every month for the years 1994-2000.

Table 7.1.1: Total/Average Number of the AAs on each month (hours 00-04).



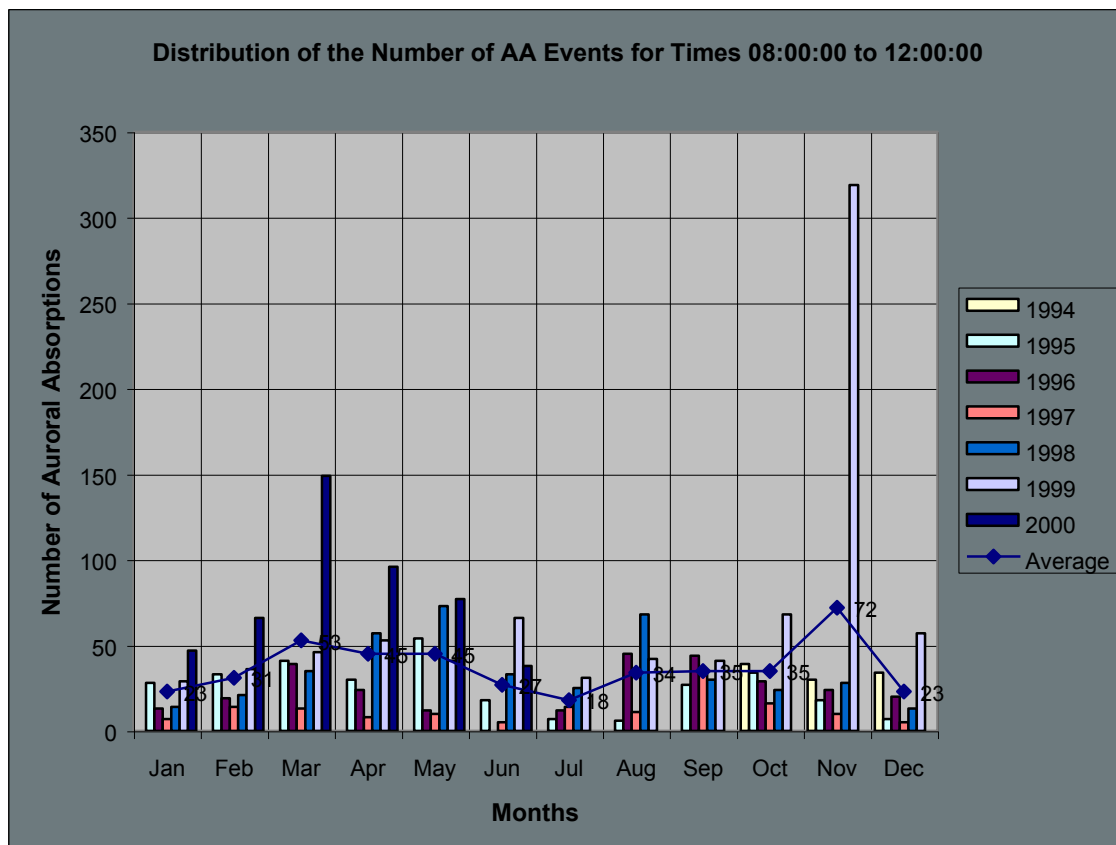
The Table 7.1.2 shows the average number of auroral absorptions that have occurred during the 04:00:00 and 08:00:00 hours on each day on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions that have occurred during the 04:00:00 and 08:00:00 hours, for every month for the years 1994-2000.

Table 7.1.2: Total/Average Number of the AAs on each month (hours 04-08).



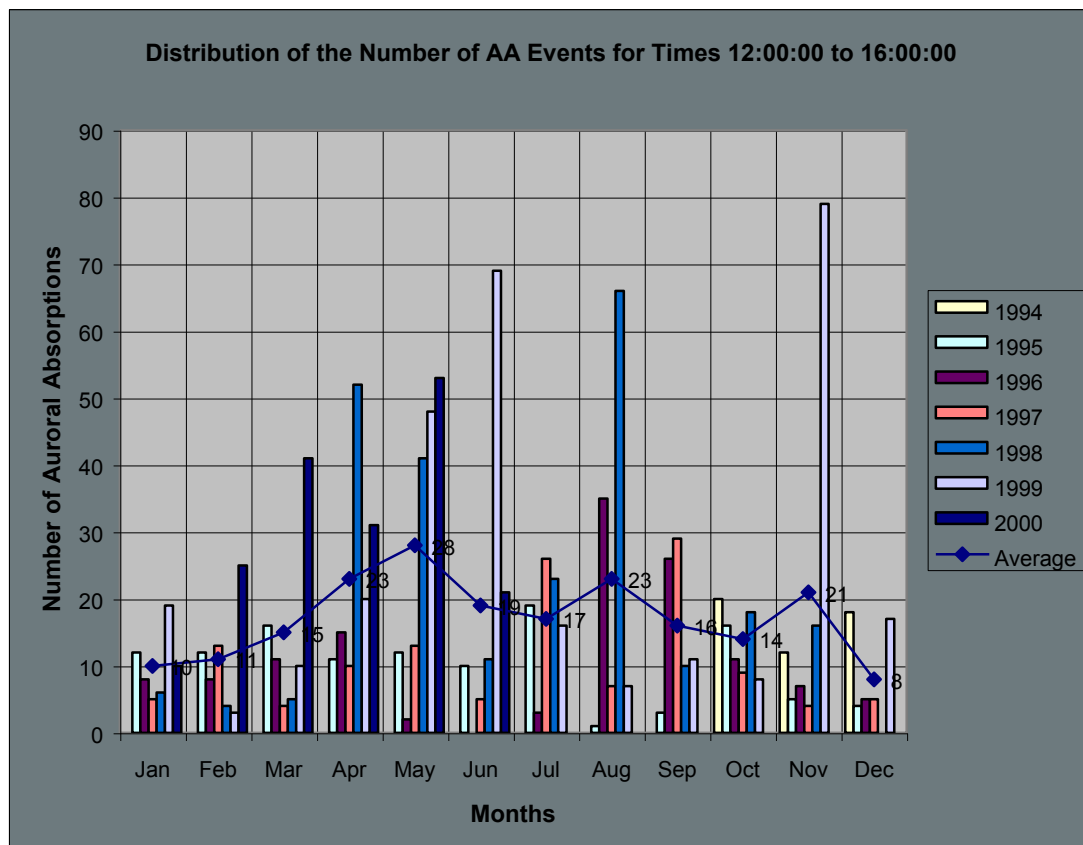
The Table 7.1.3 shows the average number of auroral absorptions that have occurred during the 08:00:00 and 12:00:00 hours on each day on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions that have occurred during the 08:00:00 and 12:00:00 hours, for every month for the years 1994-2000.

Table 7.1.3: Total/Average Number of the AAs on each month (hours 08-12).



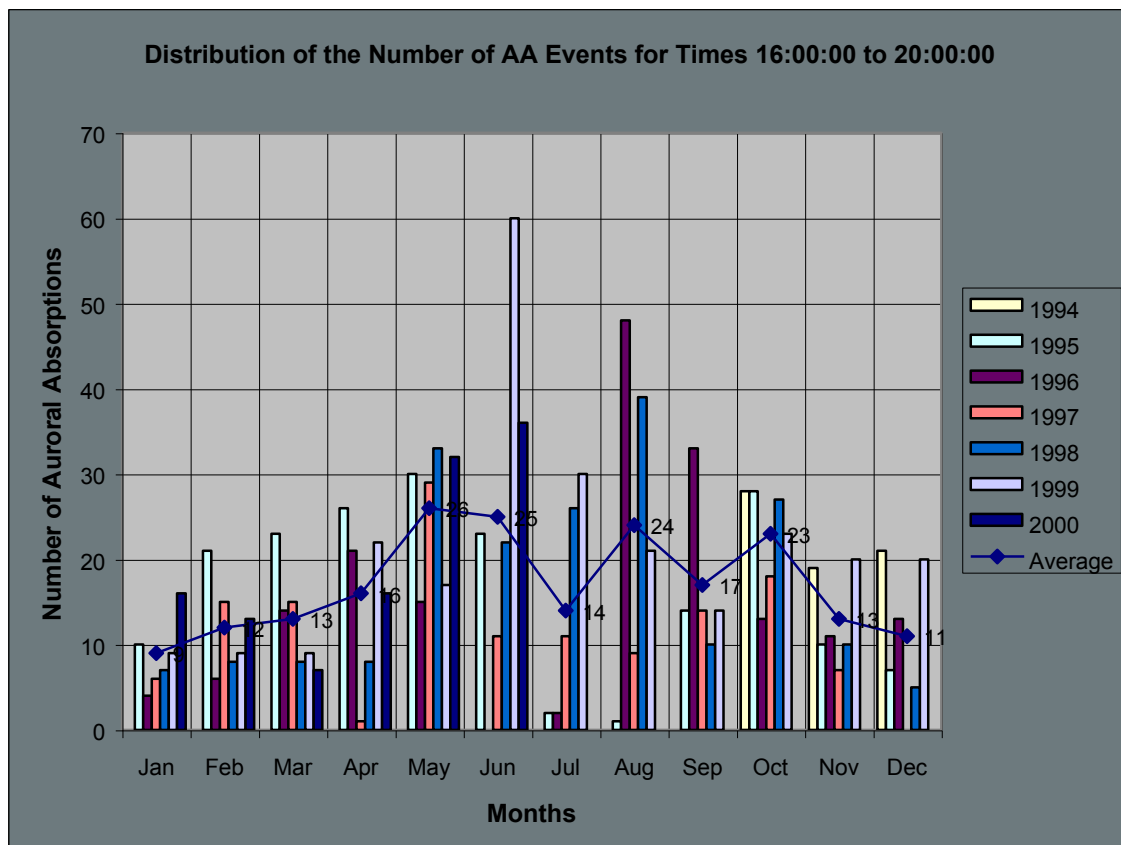
The Table 7.1.4 shows the average number of auroral absorptions that have occurred during the 12:00:00 and 16:00:00 hours on each day on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions that have occurred during the 12:00:00 and 16:00:00 hours, for every month for the years 1994-2000.

Table 7.1.4: Total/Average Number of the AAs on each month (hours 12-16).



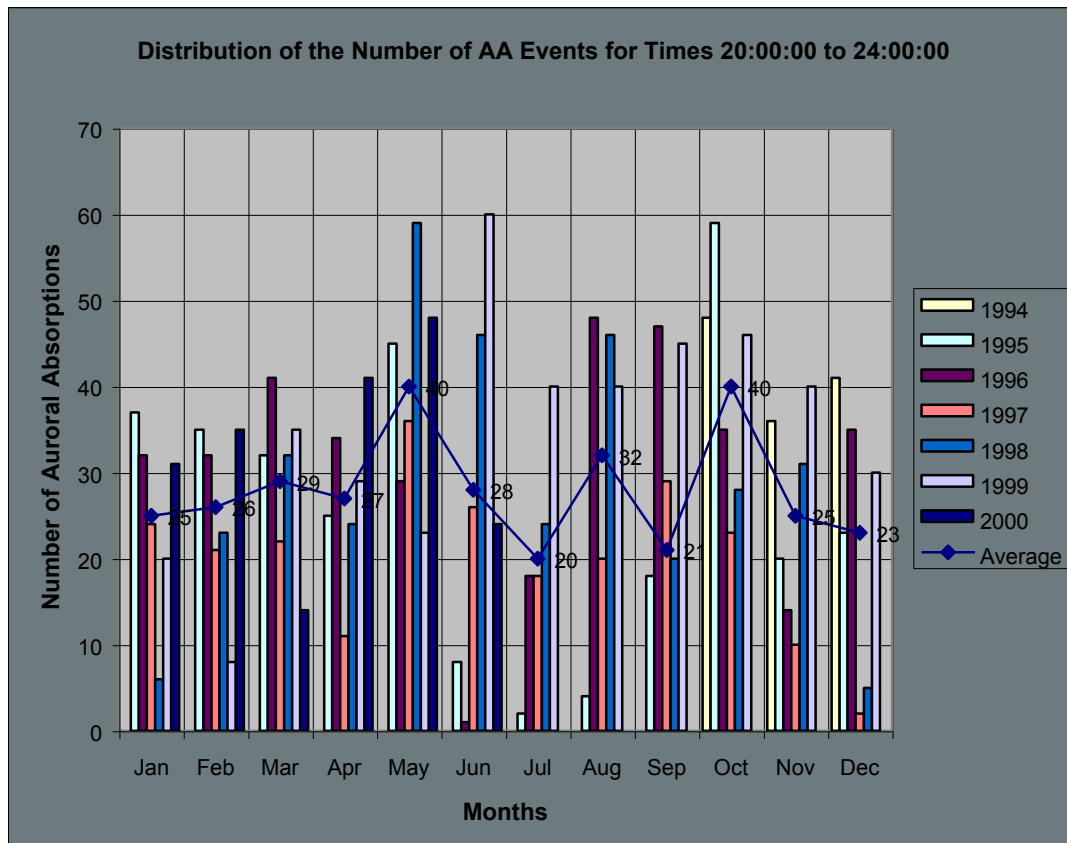
The Table 7.1.5 shows the average number of auroral absorptions that have occurred during the 16:00:00 and 20:00:00 hours on each day on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions that have occurred during the 16:00:00 and 20:00:00 hours, for every month for the years 1994-2000.

Table 7.1.5: Total/Average Number of the AAs on each month (hours 16-20).



The Table 7.1.6 shows the average number of auroral absorptions that have occurred during the 20:00:00 and 24:00:00 hours on each day on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions that have occurred during the 20:00:00 and 24:00:00 hours, for every month for the years 1994-2000.

Table 7.1.6: Total/Average Number of the AAs on each month (hours 20-24).

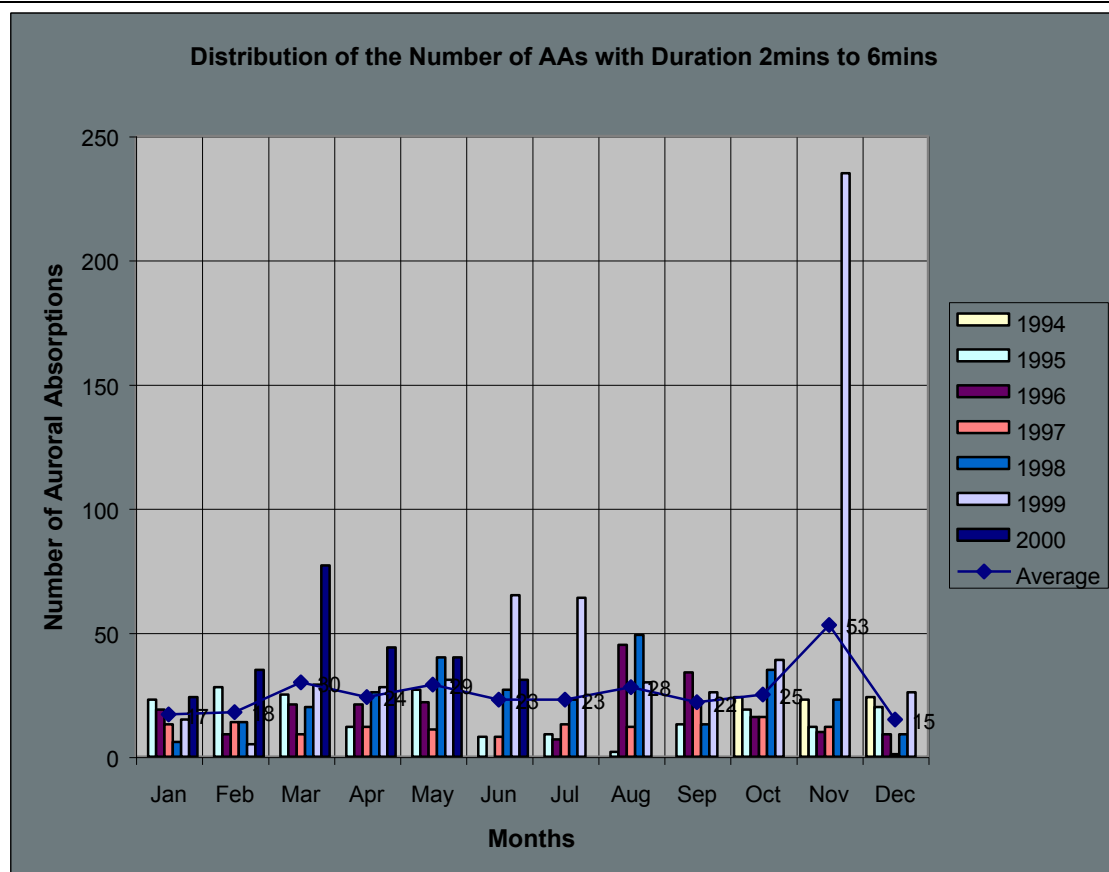


7.2 Auroral Absorption Events' Duration Distribution

Collecting all the useful information for the auroral absorptions from the Statistical Analysis results, a graph has been produced showing how the number of the auroral absorptions that depend on their duration varies on each month for every year.

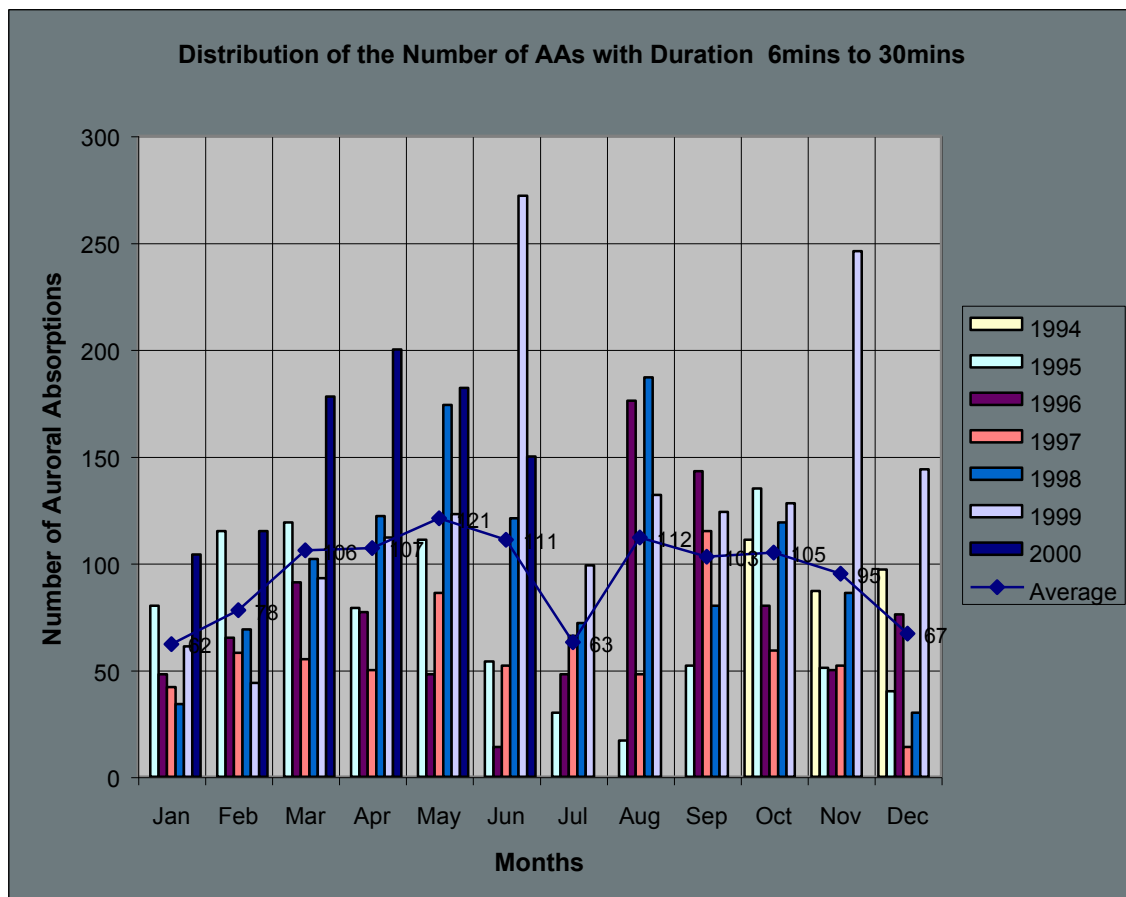
The Table 7.2.1 shows the average number of auroral absorptions that have duration between 2 minutes and 6 minutes on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions with duration between 2 minutes and 6 minutes for every month for the years 1994-2000. Note, the case that is examined now is for the spike events.

Table 7.2.1: Total/Average Number of the AAs on each month (between 2-6mins).



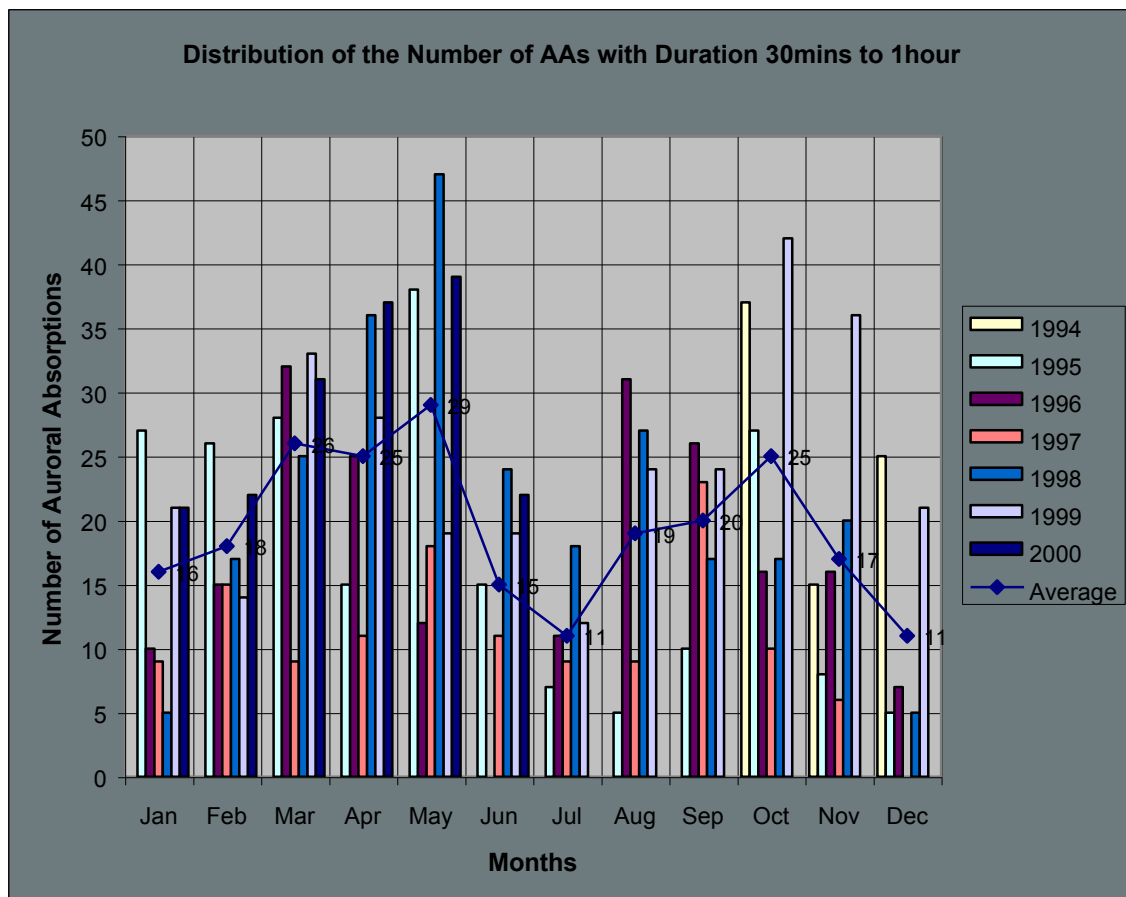
The Table 7.2.2 shows the average number of auroral absorptions that have duration between 6 minutes and 30 minutes on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions with duration between 6 minutes and 30 minutes for every month for the years 1994-2000.

Table 7.2.2: Total/Average Number of the AAs on each month (between 6-30mins).



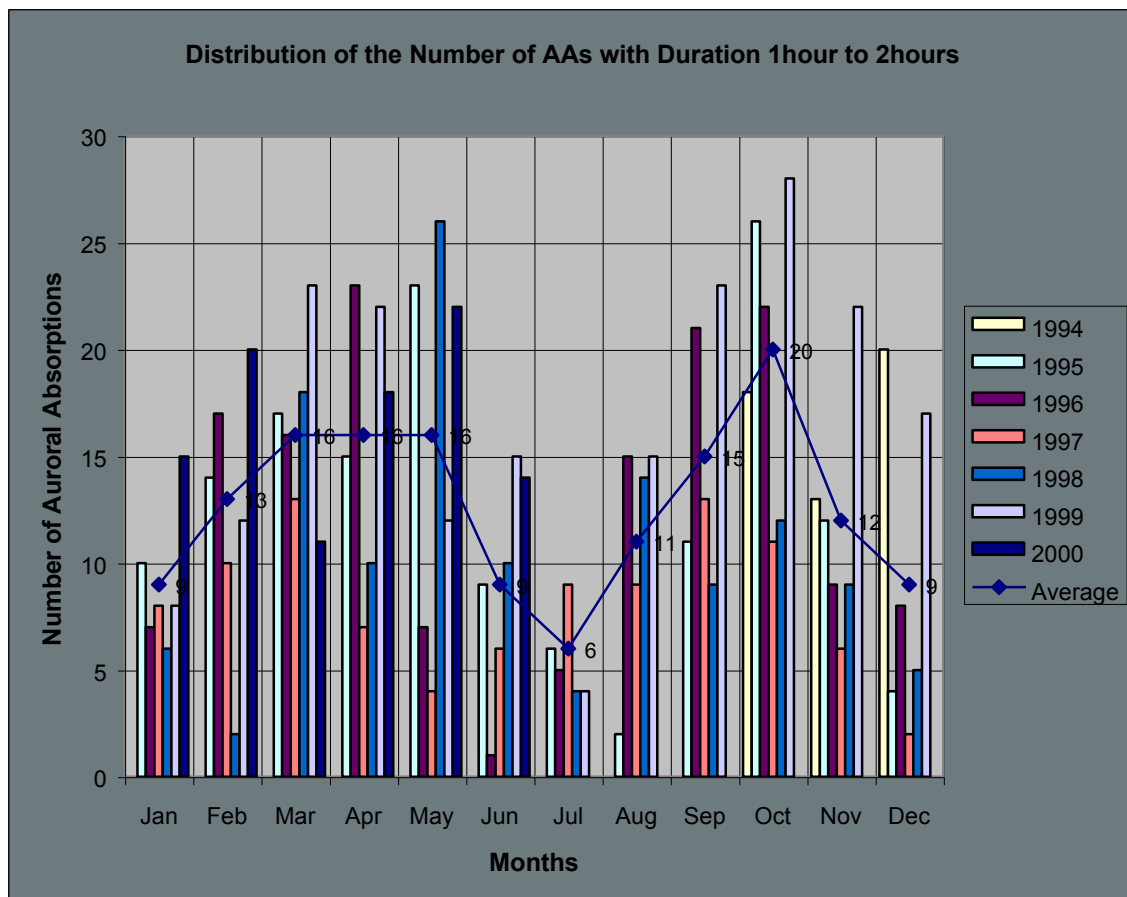
The Table 7.2.3 shows the average number of auroral absorptions that have duration between 30 minutes and 1 hour on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions with duration between 30 minutes and 1 hour for every month for the years 1994-2000.

Table 7.2.3: Total/Average Number of the AAs on each month (between 30mins-1hour).



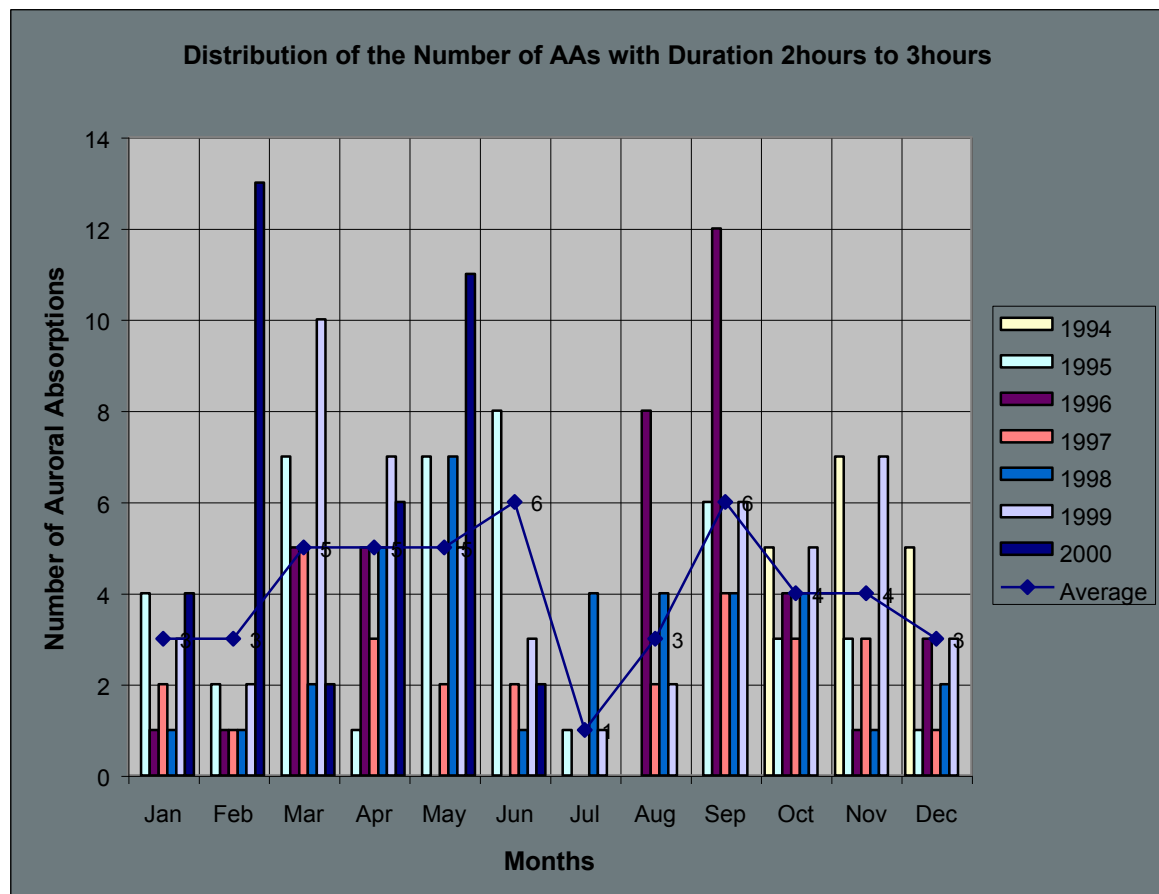
The Table 7.2.4 shows the average number of auroral absorptions that have duration between 1 hour and 2 hours on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions with duration between 1 hour and 2 hours for every month for the years 1994-2000.

Table 7.2.4: Total/Average Number of the AAs on each month (between 1-2hours).



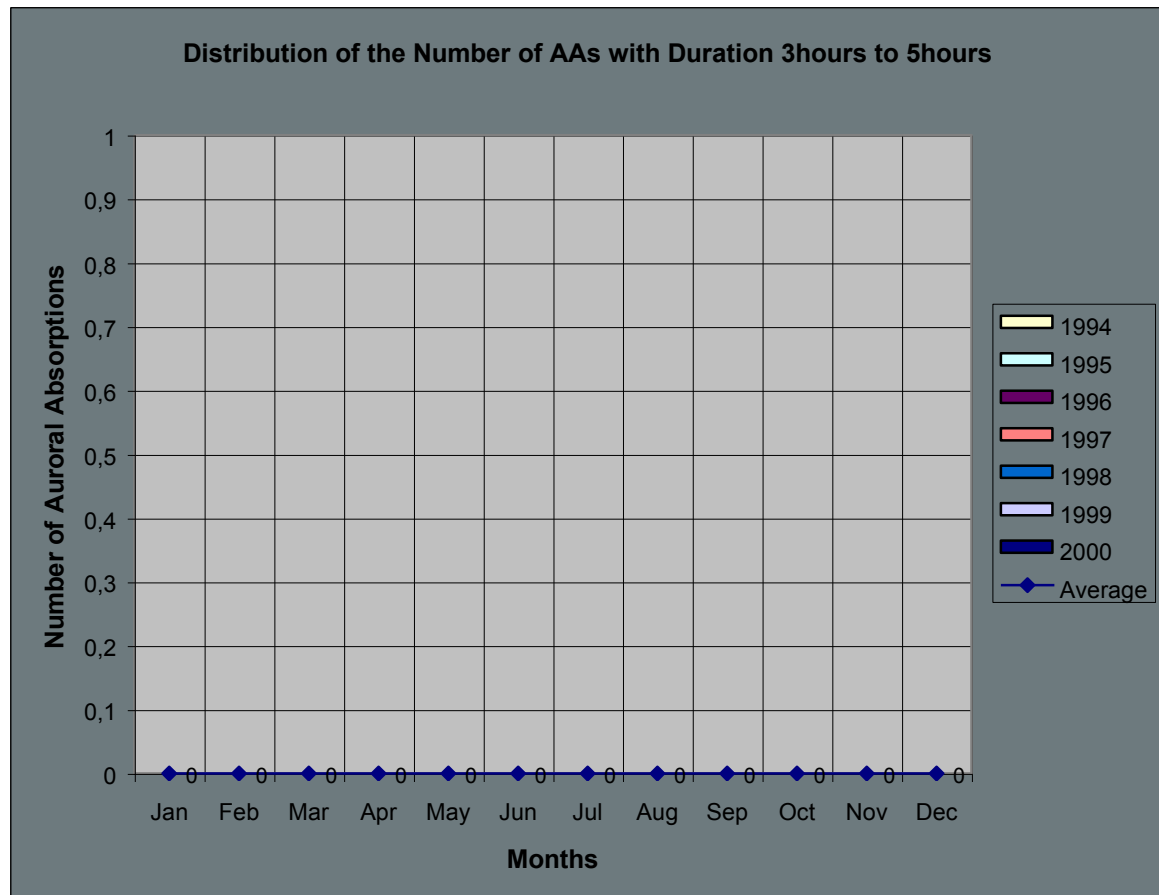
The Table 7.2.5 shows the average number of auroral absorptions that have duration between 2 hours and 3 hours on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions with duration between 2 hours and 3 hours for every month for the years 1994-2000.

Table 7.2.5: Total/Average Number of the AAs on each month (between 2-3hours).



The Table 7.2.6 shows the average number of auroral absorptions that have duration between 3 hours and 5 hours on every month (for the years 1994 to 2000); also provides us with some extra information about the number of the auroral absorptions with duration between 3 hours and 5 hours for every month for the years 1994-2000.

Table 7.2.6: Total/Average Number of the AAs on each month (between 3-5hours).



CHAPTER 8. PROJECT LIMITATIONS & FURTHER WORK

On this chapter, the project limitations are analysed and any further work that it could be done on the project is discussed. Note that all the project specifications have been completed successfully and extra work has been performed as well. Further, the project's MATLAB functions have been tested thoroughly; all the possible inputs for different iris data samples (i.e. different dates and times) have been entered into the functions and their results were examined carefully. Moreover, the same tests had been taken place on different computers with different environments and operating systems and the results were exactly the same in each case. Therefore, there are no any major or even minor limitations on the toolkit and only if some improvements need to be applied on the project then only few modifications on the MATLAB functions are necessary.

8.1 LIMITATIONS

As it has been stated above, there are no limitations on the toolkit and all the project specifications have been designed, implemented and tested successfully. Nevertheless a single minor modification has been observed and is discussed below with its solution as well.

As you may have been noticed, the MATLAB functions that fulfil the different types of the statistical analysis part (i.e. *stataa_tod.m*, *stataa_dur.m* and *statmultilong.m*) can accept only two different inputs for the variable *Beams* and these are 'widebeam' for the widebeam [50] and 'allbeams' for all the iris beams [1:50]. Although for us these two inputs were enough to complete successfully the statistical analysis, it is possible in the future a user will need to apply the statistical analysis functions on a group of the iris beams (e.g. [3 7 20:25 32]) and that group can be any combination for the iris beams.

In order for this problem to be solved, the following functions should be modified: *stataa_tod.m*, *stataa_dur.m* and *statmultilong.m*. A suggestion for the solution is to place an additional for loop inside these three functions and then the for

loop will go through only on the requested beams (i.e. let the input *Beams* is [4 10:15 22] then the for loop will be called as follows: *for analysebeams = Beams ... statements and function's body... end*; be careful to re-initialise the counters on the correct time and to modify some parts on the main body of these functions.

Note that the variable *Beams* will not take the inputs as a string (remember: 'widebeam' or 'allbeams'), but it will accept only a number or an array of numbers.

8.2 FURTHER WORK

The functions from the Data Analysis part (*irisdata_pseudoanalysis.m* and *abs_filteranalysis.m*) can recognise accurately the following events: the auroral absorptions, the spike events, the polar cap absorption, the noise and the data gaps. A further work is to design and add new rules (i.e. new event types) inside the Data Analysis functions and therefore to make the Data Analysis part more qualitative.

Furthermore, it must be noted that while an event takes place and if somewhere the time of its occurrence a data gap will occur, then the event will not be recognised correctly because we are not able to know the event's behaviour during the occurrence of a data gap and therefore there is no solution in that case. But in the case of polar cap absorption, which has duration of more than 10 hours and if a data gap with duration of less than 10 minutes will occur during the occurrence of the PCA event, then filtering is necessary to be applied and the total polar cap absorption event will be recognised correctly. An example is given to make the problem clear. Let one polar cap absorption has been recognised on the 11th of April 1995 at 02:30:00 and last for 11 hours (i.e. finished on the 11th of April 1995 at 13:30:00) and next a data gap is recognised with a start time on the 11th of April 1995 at 13:30:00 and finished on the 11th of April 1995 at 13:34:00 and finally a second polar cap absorption is recognised on the 11th of April 1995 at 13:34:00 and last for 15 hours (i.e. finished on the 12th of April 1995 at 04:34:00). That is a characteristic example where the data gap's filtering is essential. Therefore, after the filtering the correct polar cap absorption would be found with the following characteristics: started on the 11th of April 1995 at 02:30:00 and finished on the 12th of April 1995 at 04:34:00 (i.e. with duration 1 day and 2 hours and 4 minutes). Concluding, this type of filtering is important if two successive PCA events are being recognised with a data gap in

between and therefore the data gap will result in wrong recognition of two PCA events instead of one single PCA event with a longer duration.

A final extra work that may be needed is the following: Considering the project specifications, we have been asked to perform a statistical analysis on the iris data for the years 1994 to 2000. The data from the wide beam was enough for us to complete our task. But what about if later, another user will need to perform a similar or even a different statistical analysis on the iris data for the years 1994 to 2000? Although the MATLAB functions for the data and the statistical analysis have been designed, implemented and tested to work for all the iris beams, the results files (*zfilterStartYearMonthDay_EndYearMonthDay.gn*, *zcatalogueYear.gn* and *zpseudoStartYearMonthDay_EndYearMonthDay.gn*) are available only for the wide beam data. Therefore, the only that it has to be done in the future is to run the *scriptanalysis.m* function for these six years of the iris data but this time for all the fifty beams (and not only for the wide beam) and the function will load the requested iris data for all the beams from the iris server (processing time approximately 20-25 days), in parallel will analyse the loaded data and will create the results files (processing time approximately less than 10-20 minutes for all the years). After this process any user will be ready to perform a statistical analysis on all the iris beams and not only on the wide beam. It must be noted that the reason of not loading from begin the iris data for all the beams was that the loading time would be too long and there was not so much of available time.

CHAPTER 9. CONCLUSIONS

It can easily be observed that the "*Long Term Statistical Studies of Ionospheric Absorption*" project is indeed a successful Matlab5 Software Tool, which is aiming to provide great help for interpretation and analysis of the IRIS data. Completing this project and successfully achieving all the aims and specifications reported in chapter 1 can make the following conclusions:

- ✓ By examining the graph given in Table 6.4.1, it can be stated that during the months April and August the polar cap absorptions are more likely to occur having an average duration of 760 minutes (i.e. approximately 12 hours). For the years 1994-2000 no polar cap absorption has occurred in February and only one PCA have been observed in December and January. Generally, during the winter months (November to March) there are a few polar cap absorptions.
- ✓ The duration of occurrence of the PCA events increases for the years that are close to the Solar Cycle Maximum (i.e. the years 1998 to 2000) as compared with the duration of the PCA events during the Solar Cycle Minimum years (i.e. the years 1994 to 1996) as illustrated in Table 6.4.2.
- ✓ The number of the auroral absorptions is generally increased during the month periods March-May and August-November.
- ✓ The number of occurrence where auroral absorptions are less for the months of December, January, February and July, according to the results in Tables 7.1.1-7.1.6.
- ✓ The number of auroral absorptions has increased as approaching Solar Maximum (years 1998-2000).
- ✓ Most of the auroral absorptions occur during the evening and morning times (i.e. 20-24, 00-04, 04-08 and 08-12).
- ✓ Finally, for the auroral absorptions that occur during the 16:00:00 and 20:00:00 times and for the months January to May, the number of their

occurrence is constant despite on the year of the Solar Cycle (Maximum or Minimum; i.e. 1994 or 1995 or ... 2000).

- ✓ The number of auroral absorption events does not correlate with Solar Cycle in contrast to PCA events, which are Solar Cycle dependent.
- ✓ No occurrences of the auroral absorptions with duration 3-5 hours have been observed according to the results in Tables 7.2.1-7.2.6.
- ✓ Conversely, month periods January-February, July and December can be characterised as quite periods.
- ✓ The number of the auroral absorptions with duration 6 to 30 minutes exceeds the one hundred for the months of March-June and August-October.
- ✓ The number of the auroral absorptions with duration 30 minutes to 1 hour is less than the number of the auroral absorptions with duration 6 to 30 minutes (for every month of all the six years); but their monthly distribution of both types of auroral absorptions is similar.
- ✓ Every mnth the number of the auroral absorptions with duration 1 to 2 hours is increased rapidly from the year 1994 to 2000. Furthermore, the months March-May and September-October the average number of the auroral absorptions overtops the 15, compared with the months January, June, July and December where the average number of the auroral absorptions is less than 10.
- ✓ Also the number of the auroral absorptions with duration 2 to 3 hours does not exceed the 3 to 5 occurrences per month.

Finally, it worth to mention what it has been gained by this project:

- ✓ Depth study of the characteristics of all the phenomena that can be seen in the IRIS data (i.e. auroral absorptions, spike events, polar cap absorptions, noise and data gaps).
- ✓ Obtained experience on analysing a large amount of data fast and without to overload the server's memory and CPU.
- ✓ Obtained experience on long-term statistical studies.
- ✓ Obtained experience on designing and implementing a database in MATLAB5.
- ✓ Became experts on MATLAB5 programming.
- ✓ Learned to design efficient and complex algorithms.

- ✓ Learned to implement reliable and fast MATLAB5 functions.
- ✓ Using the results that obtained from the long-term statistical analyses, we had the opportunity to study them, analyse them and from their analysis we were able to find the distribution and the behaviour of the polar cap absorptions duration and of the number of the auroral absorptions that depend on their duration and on the time of the day as well.

An important issue is that this toolkit has been designed and implemented as a useful tool for the IRIS system, but a special characteristic that makes the software very powerful is that can be used for any type of data interpretation and for any type of statistical analysis as well. Due to efficient algorithms design, by modifying only few parts of the source code (not the algorithms) and by changing the function rules in terms of the new type of data, the toolkit will be transformed in a new software that will serve the new demands for the new type of data.

REFERENCES

- [1] J. K. Hargreaves. 'The solar-terrestrial environment' Cambridge Atmospheric and Space Sciences Series, First Published 1992.
- [2] Steven T. Suess and Bruce T. Tsurutani. 'From the Sun: Auroras, Magnetic Storms, Solar Flares, Cosmic Rays' American Geophysical Union. Washington, DC, 1998.
- [3] P. N. Collis and J. K. Hargreaves. Coordinated Studies Using Imaging Riometer and Incoherent Scatter Radar. *Journal of Atmospheric and Solar-Terrestrial Physics*, volume 59(8):pp. 873-890, 1997.
- [4] P. N. Collis, J. K. Hargreaves, W. G. Howarth and G. P. White. Joint Imaging Riometer – Incoherent Scatter Radar Observations: A Four-dimensional Perspective on Energetic Particle Input to the Auroral Mesosphere. *Advances in Space Research*, volume 20(6):pp. 1165-1168, 1997.
- [5] P. N. Collis, J. K. Hargreaves and G. P. White. A Localised Co-rotating Auroral Absorption Event Observed Near Noon Using Imaging Riometer and EISCAT. *Annales Geophysicae – Atmospheres, Hydrospheres and Space Sciences*, volume 14(12):pp. 1305-1316, 1997.
- [6] J. K. Hargreaves, S. Browne, H. Ranta, A. Ranta, T. J. Rosenberg and D. L. Detrick. A Study of Substorm-associated Night-side Spike Events in Auroral Absorption Using Imaging Riometers at the South Pole and Kilpisjarvi. *Journal of Atmospheric and Solar-Terrestrial Physics*, volume 59(8):pp. 853-872, 1997.
- [7] H. Ranta, A. Ranta, J. K. Hargreaves and S. Browne. Localised Absorption Events in the Afternoon Sector. *Journal of Atmospheric and Solar-Terrestrial Physics*, volume 59(8):pp. 891-902, 1997.

- [8] Duane Hanselman and Bruce Littlefield. 'Mastering MATLAB5, A Comprehensive Tutorial and Reference' Prentice Hall, 1998.
- [9] Duane Hanselman and Bruce Littlefield (by MathWorks). 'MATLAB version 5 User's Guide' Prentice Hall, 1997.
- [10] MATLAB 5.3 Software Package for UNIX Environment. 'MATLAB Help Desk' MathWorks, 1998.
- [11] MATLAB Help Desk
URL: <http://iris.lancs.ac.uk/matlab/>
- [12] WWW MATLAB Help Viewer
URL: <http://www.dcs.lancs.ac.uk/marple/cgi-bin/wmathelp>
- [13] IRIS Toolkit
URL: <http://www.dcs.lancs.ac.uk/iono/iris/iristool/>
- [14] IRIS Toolkit Classes
URL: <http://www.dcs.lancs.ac.uk/iono/iris/iristool/classes/>
- [15] Lancaster University, Ionosphere and Radio Propagation Group
URL: <http://www.dcs.lancs.ac.uk/iono/>
- [16] IRIS – Imaging Riometer for Ionospheric Studies
URL: <http://www.dcs.lancs.ac.uk/iono/iris/>
- [17] Kilpisjarvi IRIS On Line Data
URL: <http://www.dcs.lancs.ac.uk/iono/iris/data/>
- [18] Science Direct
URL: <http://www.sciencedirect.com>

[19] NASA Education

URL: <http://www-istp.gsfc.nasa.gov/Education/>

[20] NASA Glossary

URL: <http://www-spof.gsfc.nasa.gov/Education/gloss.html>

[21] NOAA Education and Outreach

URL: <http://www.sel.noaa.gov/info/>

APPENDICES

APPENDIX A - MATLAB FUNCTIONS

APPENDIX B - STATISTICAL ANALYSIS RESULTS (TABLES)

APPENDIX C - STATISTICAL ANALYSIS RESULTS (HISTOGRAMS)

APPENDIX A

LONG TERM STATISTICAL STUDIES OF IONOSPHERIC ABSORPTION MATLAB FUNCTIONS

scriptanalysis.m

```
function scriptanalysis(year, loadmonths);

%SCRIPTANALYSIS
%
%Date Started: 18/06/2000 9:39pm
%Date Completed: 18/06/2000 9:45pm
%Creator: G.Nikitas
%DCS, Lancaster University
%
% run this script to make the analysis
% for one year (12 files, for each month).
% this function takes as an input the year,
% and for the specific year
% if you want to make the analysis for few
% months only, enter the year and the months.
% e.g. scriptanalysis(1995, [2 5 8:11]).

% set the default value for the monthss.
if nargin < 2
    loadmonths = [1:12];
end

% take the time the processing startss.
processyearstarts = time;

% start from the first month (1 - Jan) and
% finish at the last moth (12 - Dec).
for month = loadmonths

% take the time the processing starts.
processmonthstarts = time;
```

```
% print a message on the workspace.
fprintf('\n\nStarting Data Analysis');
fprintf('\nYear %d, Month %d\n', year, month);

% depending on the inputs given by the user,
% initialise these two variables.
next_month = month + 1;
next_year = year + 1;

data_duration = timespan(1, 'h');      % data duration for 1
day.
data_resolution = timespan(1, 'm');    % data resolution for 1
minute.

% these are for testing.
beams_wide = [50];                    % only the widebeam.
beams_some = [4 7 42 50];             % some of the beams.
beams_all = [1:50];                   % all the beams.

beams_requested = beams_wide;          % beams requested.

% if the month you entered is between 1 and 11
% then execute this if statement.
if ((month == 1 | month == 2 | month == 3 | month == 4 | ...
    month == 5 | month == 6 | month == 7 | month == 8 | ...
    month == 9 | month == 10 | month == 11) ...
    & (year >= 1994))

% specify the general characteristics of the data.
data_starttime = ...
    time([year month 01 0 0 0]);      % start date - to load
the data.
data_endtime = ...
    time([year next_month 01 0 0 0]); % end date - to load
the data.
```

```
% call the irisdata_pseudoanalysis function.
irisdata_pseudoanalysis(data_starttime, data_endtime, ...
    data_duration, data_resolution, beams_requested);

% if the month you entered is between 12
% then execute this elseif statement.
elseif (month == 12 & year >= 1994)

    % specify the general characteristics of the data.
data_starttime = ...
    time([year month 01 0 0 0]);    % start date - to load the
data.
data_endtime = ...
    time([next_year 01 01 0 0 0]); % end date - to load the
data.

% call the irisdata_pseudoanalysis function.
irisdata_pseudoanalysis(data_starttime, data_endtime, ...
    data_duration, data_resolution, beams_requested);

% if the month you entered is not between 1 to 11 or 12
% then execute this statement.
else

    error('Invalid entry for the month or for the year');

end % end of if-elseif-else statement.

% the name of the file where the results of the
% pseudo analysis are going to be saved,
% depends on start and end times of the load data.

% this information is needed on the abs_filteranalysis
% function...
```

```
absresults_pseudo = ...
    char(['zpseudo' strftime(data_starttime, '%Y%m%d') ...
        '_' strftime(data_endtime, '%Y%m%d') '.gn']);

% call the abs_filteranalysis function.
abs_filteranalysis(absresults_pseudo);

% take the time the processing ends.
processmonthends = time;

% find the total processing time.
processmonthtime = processmonthends - processmonthstarts;

% print a message on the workspace.
fprintf('\nProcessing Time (Year %d, Month %d) : %s days and
%s:%s:%s hours\n', ...
    year, month, ...
    strftime(processmonthtime, '%d'),
    strftime(processmonthtime, '%H'), ...
    strftime(processmonthtime, '%M'),
    strftime(processmonthtime, '%S'));

end % and of for loop for the months.

% take the time the processing ends.
processyearends = time;

% find the total processing time.
processyeartime = processyearends - processyearstarts;

% print a message on the workspace.
fprintf('\n\nTotal Processing Time (Year %d) : %s days and
%s:%s:%s hours\n', ...
```

```
year, ...  
    strftime(processyeartime, '%d'), strftime(processyeartime,  
    '%H'), ...  
    strftime(processyeartime, '%M'), strftime(processyeartime,  
    '%S'));
```

irisdata_pseudoanalysis.m

```
function irisdata_pseudoanalysis(data_starttime, ...
    data_endtime, data_duration, data_resolution,...
    beams_requested);

%IRISDATA_PSEUDOANALYSIS
%
%Date Started: 26/05/2000 5:48pm
%Date Completed: 27/05/2000 3:41am
%Creator: G.Nikitas
%DCS, Lancaster University
%
% pseudo analysis for absorption, noise and data gaps
% on the iris data.
% loads the data every hour until the requested end time.
% take the first requested beam - analyse its data.
% take the next requested beam - analyse its data.
% repeat until to analyse and the last requested beam.
% save the results of the pseudo analysis in a file.
% note that if the file of the iris data does not exists, the
% program generates automatically one raw of NaNs for every
hour of
% missing data.
% if the user wishes to terminate (kill) the process the
program
% will accept it and will exit to Matlab Workspace.

% if the variables below have not be given by the user,
% then set their defaults values.

% default value for the beams is the widebeam.
if nargin < 5, beams_requested = 50;
% default value for the resolution is 1 minute.
elseif nargin < 4, data_resolution = timespan(1, 'm');
% default value for the duration of the data to be load is 1
```



```
hour.
elseif nargin < 3, data_duration = timespan(1, 'h');
% default values for the start and end times do not exist.
% return an error message and exit.
elseif nargin < 2, error('Inefficient number of inputs!');
return;
end

% go through all the requested beams
% i.e. once the analysis of one beam (for the total duration)
% is finished, take the next beam and repeat the analysis
% process, and so on until the last beam.
for beams_available = beams_requested

% print a message on the workspace.
fprintf('\n\nProcessing Beam %d\n', beams_available);

% initialise a variable to load the data every hour.
% it will be used on the while loop below.
data_currenttime = data_starttime;

% load the data every hour until the end time.
while data_currenttime < data_endtime

% the name of the file where the results of the
% pseudo analysis are going to be saved,
% depends on start and end times of the load data.
absresults_pseudo = ...
    char(['zpseudo' strftime(data_starttime, '%Y%m%d') ...
        '_' strftime(data_endtime, '%Y%m%d') '.gn']);

% open a file to save the results (append mode).
% the results are saved in ASCII format.
fid_pseudo = fopen(absresults_pseudo, 'a');
```

```
% when the data file exist on the IRIS Server.
try

% call the function that loads the requested data.
% it returns useful information about the loaded data.
[data_values, current_beam, current_starttime, ...
    current_endtime, current_duration, current_datasize] ...
= irisdata_load(data_currenttime, data_duration, ...
    data_resolution, beams_available);

% for the current loaded data, the maximum duration
% of absorption or datagap event
% can be the length of the loaded data.
abs_data = zeros(1,length(data_values));
datagap_data = zeros(1,length(data_values));
noise_data = zeros(1,length(data_values));

% initialise a counter that will keep the
% position of the data in the data values matrix.
data_values_counter = 1;

% start from the begin of the data values matrix and
% stop at the end of the data values matrix.
while data_values_counter <= length(data_values)

    % the minimum value for absorption is 0.5 dB.
    abs_limit = 0.5;

    % the minimum value for noise is -1 dB.
    noise_limit = -1;

    % if a value for the data is greater than 0.5dB
    % then absorption occurs.
    if (data_values(data_values_counter)) >= abs_limit
```

```
% take the start time of the absorption event.
abs_startsample = data_values_counter - 1;
abs_starttime = data_currenttime + ...
abs_startsample * data_resolution;

% abs_counter takes the value of the
% data_values_counter when absorption occurs.
abs_counter = data_values_counter;

% special case the event to occur at the last minute
before
% the end of an hour.
abs_data(abs_counter) = data_values(abs_counter);

% while absorption occurs, store
% the (absorption) data in a matrix.
while (data_values(abs_counter) >= abs_limit) ...
    & abs_counter < length(data_values)

% store the absorption data in abs_data matrix.
    abs_data(abs_counter) = data_values(abs_counter);

% go to the next sample of the data_values.
abs_counter = abs_counter + 1;
end % end of while loop where the absorption data are
stored.

% reinitialise the counter for the datavalues with its
new value.
% this is needed because the 2nd while loop has already
read
% an amount of data.
% therefore the counter should take the value fo the
% latest sample of data that has been analysed so far.
data_values_counter = abs_counter;
```

```
% the matrix abs_data contains the current absorption
event

% but at the begin it has been initialised with zeros
% equal the length of the loaded data.
% so now remove the remaining zeros from the end of the
% matrix and a new matrix is created 'abs_event', which
% keeps only the data of the absorptigon event.
abs_event = nonzeros(abs_data);

% find the maximum and mean values
% of the current absorption event.
max_abs = max(abs_event);
mean_abs = mean(abs_event);

% take the end time of the absorption event.
abs_endsample = data_values_counter;
abs_endtime = data_currenttime + ...
abs_endsample * data_resolution;

% find the duration of the absorption event.
abs_duration = ...
abs_endtime - abs_starttime;

% return the results of the pseudo analysis
% for the Absorption occurences.

% column 1      : 1 - means Absorption.
% column 2      : 1 to 50 - the beam that is currently
analysed.
% column 3-8    : hours minutes seconds day month year,
start time.
% column 9-12   : hours minutes seconds day month year,
end time.
% column 13-18: days hours minutes seconds, duration.
% column 19     : max value of the absorption in dBs.
% column 20     : mean value of the absorption in dBs.
```

```

    fprintf(fid_pseudo, '%d %d %s %s %s %.2f %.2f\n', ...
1, current_beam, strftime(abs_starttime, ...
'%Y %m %d %H %M %S'), strftime(abs_endtime, ...
'%Y %m %d %H %M %S'), strftime(abs_duration, ...
'%d %H %M %S'), max_abs, mean_abs);

% reinitialise to zero some of the counters/variables.
abs_counter = 0;
abs_data = zeros(1, length(data_values));
abs_startsample = 0;
abs_endsample = 0;
end % end of if statement that checks for absorption.

% if a value for the data is NaN
% then data gap occurs.
if isnan(data_values(data_values_counter))

    % take the start time of the data gap event.
    datagap_startsample = data_values_counter - 1;
    datagap_starttime = data_currenttime + ...
    datagap_startsample * data_resolution;

    % datagap_counter takes the value of the
    % data_values_counter when data gap occurs.
    datagap_counter = data_values_counter;

    % special case the event to occur at the last minute
before
    % the end of an hour.
    datagap_data(datagap_counter) =
data_values(datagap_counter);

    % while data gap occurs, store
    % the (data gap) data in a matrix.

```

```
while isnan(data_values(datagap_counter)) ...
    & (datagap_counter < length(data_values))

    % store the datagap data in the datagap_data matrix.
    datagap_data(datagap_counter) = ...
    data_values(datagap_counter);

% go to the next sample of the data_values.
    datagap_counter = datagap_counter + 1;
end % end of while loop where the datagap data are
stored.

% reinitialise the counter for the datavalues with its
new value.
% this is needed because the 2nd while loop has already
read
% an amount of data.
% therefore the counter should take the value fo the
% latest sample of data that has been analysed so far.
data_values_counter = datagap_counter;

% the matrix datagap_data contains the current datagap
event
% but at the begin it has been initialised with zeros
% equal the length of the loaded data.
% so now remove the remaining zeros from the end of the
% matrix and a new matrix is created 'datagap_event',
which
% keeps only the data of the datagap event.
datagap_event = nonzeros(datagap_data);

% find the maximum and mean values
% of the current datagap event.
max_datagap = nan; % NaN expected.
mean_datagap = nan; % NaN expected.

% take the end time of the datagap event.
```

```
    datagap_endsample = data_values_counter;
    datagap_endtime = data_currenttime + ...
    datagap_endsample * data_resolution;

    % find the duration of the datagap event.
    datagap_duration = ...
    datagap_endtime - datagap_starttime;

    % return the results of the pseudo analysis
    % for the Data Gap occurrences.

    % column 1      : 2 - means Data Gap.
    % column 2      : 1 to 50 - the beam that is currently
analysed.
    % column 3-8    : hours minutes seconds day month year,
start time.
    % column 9-12   : hours minutes seconds day month year,
end time.
    % column 13-18: days hours minutes seconds, duration.
    % column 19     : max value of the absorption in dBs.
    % column 20     : mean value of the absorption in dBs.

    fprintf(fid_pseudo, '%d %d %s %s %s %.2f %.2f\n', ...
2, current_beam, strftime(datagap_starttime, ...
'%Y %m %d %H %M %S'), strftime(datagap_endtime, ...
'%Y %m %d %H %M %S'), strftime(datagap_duration, ...
'%d %H %M %S'), max_datagap, mean_datagap);

    % reinitialise to zero some of the counters/variables.
    datagap_counter = 0;
    datagap_data = zeros(1, length(data_values));
    datagap_startsample = 0;
    datagap_endsample = 0;

end % end of if statement that checks for data gaps.
```

```
% if a value for the data is less than -1 dB
% then noise occurs.
if (data_values(data_values_counter)) <= noise_limit

    % take the start time of the noise event.
    noise_startsample = data_values_counter - 1;
    noise_starttime = data_currenttime + ...
noise_startsample * data_resolution;

    % noise_counter takes the value of the
    % data_values_counter when noise occurs.
    noise_counter = data_values_counter;

    % special case the event to occur at the last minute
before
    % the end of an hour.
    noise_data(noise_counter) = data_values(noise_counter);

    % while noise occurs, store
    % the (noise) data in a matrix.
    while (data_values(noise_counter) <= noise_limit) ...
        & noise_counter < length(data_values)

        % store the noise data in abs_data matrix.
        noise_data(noise_counter) =
data_values(noise_counter);

        % go to the next sample of the data_values.
        noise_counter = noise_counter + 1;
    end % end of while loop where the noise data are stored.

    % reinitialise the counter for the datavalues with its
new value.
    % this is needed because the 2nd while loop has already
read
```



```
% an amount of data.
% therefore the counter should take the value fo the
% latest sample of data that has been analysed so far.
data_values_counter = noise_counter;

% the matrix noise_data contains the current noise event
% but at the begin it has been initialised with zeros
% equal the length of the loaded data.
% so now remove the remaining zeros from the end of the
% matrix and a new matrix is created 'noise_event',
which
% keeps only the data of the noise event.
noise_event = nonzeros(noise_data);

% find the minimum and mean values
% of the current noise event.
% for the max value for noise (-ve values)
% we should find the minimum.
max_noise = min(noise_event);
mean_noise = mean(noise_event);

% take the end time of the noise event.
noise_endsample = data_values_counter;
noise_endtime = data_currenttime + ...
noise_endsample * data_resolution;

% find the duration of the noise event.
noise_duration = ...
noise_endtime - noise_starttime;

% return the results of the pseudo analysis
% for the Noise occurences.

% column 1      : 3 - means Noise.
% column 2      : 1 to 50 - the beam that is currently
analysed.
```

```
% column 3-8 : hours minutes seconds day month year,
start time.
% column 9-12 : hours minutes seconds day month year,
end time.
% column 13-18: days hours minutes seconds, duration.
% column 19 : max value of the absorption in dBs.
% column 20 : mean value of the absorption in dBs.

fprintf(fid_pseudo,'%d %d %s %s %s %.2f %.2f\n', ...
3, current_beam, strftime(noise_starttime, ...
'%Y %m %d %H %M %S'), strftime(noise_endtime, ...
'%Y %m %d %H %M %S'), strftime(noise_duration, ...
'%d %H %M %S'), max_noise, mean_noise);

% reinitialise to zero some of the counters/variables.
noise_counter = 0;
noise_data = zeros(1,length(data_values));
noise_startsample = 0;
noise_endsample = 0;
end % end of if statement that checks for noise.

% go to the next sample of the data_values_counter.
data_values_counter=data_values_counter + 1;
end % end of while loop for the data of the current beam.

% when the data file does NOT exist on the IRIS Server.
catch

killtype1 = 'Interrupt';
killtype2 = 'Terminate';

% if error occurs,
% check if it is (the error) because the file of iris
```

```
% absorption data does not occurs
% or
% because the user requested to terminate the program
% i.e. press Ctrl-C or kill the process
if strcmp(lasterr, killtype1) | strcmp(lasterr, killtype2)

    % print a message on the Matlab Workspace.
    fprintf('\n\nProcess Terminated by the User!\n\n');

    % exit the process and return to Matlan.
    return;

end % end of if statement for the users request on
terminating
    % the program.


    % printf only NaNs if the file for the specific hour
(file)
    % does not exists.
    fprintf(fid_pseudo, ...
'NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
NaN NaN NaN NaN NaN\n');


    % find the missing file.
    invalidfile = char([strftime(data_currenttime, '%d/%m/%Y
%H:%M:%S')]);


    % printf a message on the workspace.
    fprintf('\n\nData File Does NOT exists. Date %s\n\n',
invalidfile);

end % end of try-catch statement for the data file.


% close the file that has been opened to save the results.
```

```

status = fclose(fid_pseudo);

% once one hour of data has been analysed,
% go to the next hour of data.
data_currenttime = data_currenttime + timespan(1, 'h');
end % end of while loop for the total data.

end % end of for loop for all the available beams.

% open the file of the pseudo analysis for second time,
% to add the extra line (with zeros).
fid_pseudo = fopen(absresults_pseudo, 'a');

% once finished the pseudo analysis,
% add a line of numbers (only zeros) that will
% specify the end of file.
fprintf(fid_pseudo, ...
        '%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d\n', ...
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0);

% close the file that has been opened for second time,
% to add the extra line (with zeros).
status = fclose(fid_pseudo);

```

abs_filteranalysis.m

```
function abs_filteranalysis(absresults_pseudo);

%ABS_FILTERANALYSIS
%
%Date Started: 28/05/2000 5:48pm
%Date Completed: 14/06/2000 2:25pm
%Creator: G.Nikitas
%DCS, Lancaster University
%
% filter analysis for absorption on the iris data.
% loads the results of the requested pseudo analysis.
% try to find the absorption events that have been
% seen as a two or more events and compine them into
% a one single absorption event
% (e.g 1st event from 12:46 to 12:59 and
% 2nd event from 13:00 to 13:22, that is
% a single evet from 12:46 to 13:22.
% save the results of the filter analysis in a file.

% load the requested file.
abs_pseudo = load([absresults_pseudo]);

% find the size of the abs_pseudo matrix.
abs_pseudo_size = size(abs_pseudo);

% the name of the file where the results of the
% filter analysis are going to be saved,
% depends on the filename of the
% results for the pseudo analysis.
absresults_filter = ...
    char(['zfilter' ...
        absresults_pseudo(8:length(absresults_pseudo))]);

% open a file to save the results of the filter analysis.
% the results are saved in ASCII format.
```

```
fid_filter = fopen(absresults_filter, 'a');

% initialise the matrix where the results of the
% filter analysis will be stored each time it loops
% the while loop below.
abs_filter = zeros(1,abs_pseudo_size(2));

% initialise a counter for the while loop below.
abs_pseudo_occurrence = 1;

% start from the begin of the pseudo results until
% the last pseudo absorption occurrence.
while abs_pseudo_occurrence < (abs_pseudo_size(1))

    % initialise a variable for the type of the event
    % let be general absorption.
    abs_event = 1;

    % initialise a counter for the filter absorption events.
    abs_filter_counter = abs_pseudo_occurrence;

    % if the conditions below are true
    % then pseudo absorption event has occurred.
    %
    % condition 1: it should be an absorption event (i.e. '1'),
    %               1 for absorption, 2 for data gap.
    % condition 2: the end time of a specific absorption event,
    %               must be date(Y M d) time(h 00 s).
    % condition 3: the start time of the next absorption event,
    %               must be date(Y M d) time(h 00 s).
    % condition 4: check that the absorption events have
    %               occurred on the same beam.
    if abs_pseudo(abs_filter_counter, 1) == 1 ...
        & abs_pseudo(abs_filter_counter, 13) == 0 ...
        & abs_pseudo(abs_filter_counter + 1, 7) == 0 ...
        & (abs_pseudo(abs_filter_counter, 1) == ...
            abs_pseudo(abs_filter_counter + 1, 1)) ...
```

```
& (abs_pseudo(abs_filter_counter, 2) == ...
abs_pseudo(abs_filter_counter + 1, 2))

% initialise a variable, it will be used
% for indexing in the while loop below.
abs_index = 1;

% use a while loop to find the number of pseudo
% absorption events have occurred simultaneously.
while abs_pseudo(abs_filter_counter, 13) == 0 ...
& abs_pseudo(abs_filter_counter + 1, 7) == 0 ...
& (abs_pseudo(abs_filter_counter, 1) == ...
abs_pseudo(abs_filter_counter + 1, 1)) ...
& (abs_pseudo(abs_filter_counter, 2) == ...
abs_pseudo(abs_filter_counter + 1, 2))

% find the event.
% this is still the non-analysed absorption event,
% but is used for initialising the variable.
abs_filter(1, 1) = abs_pseudo(abs_filter_counter, 1);

% find the beam.
abs_filter(1, 2) = abs_pseudo(abs_filter_counter, 2);

% find the start time.
abs_filter(1, 3) = abs_pseudo(abs_pseudo_occurrence, 3);
abs_filter(1, 4) = abs_pseudo(abs_pseudo_occurrence, 4);
abs_filter(1, 5) = abs_pseudo(abs_pseudo_occurrence, 5);
abs_filter(1, 6) = abs_pseudo(abs_pseudo_occurrence, 6);
abs_filter(1, 7) = abs_pseudo(abs_pseudo_occurrence, 7);
abs_filter(1, 8) = abs_pseudo(abs_pseudo_occurrence, 8);

% find the end time.
abs_filter(1, 9) = abs_pseudo(abs_filter_counter + 1, 9);
abs_filter(1,10) = abs_pseudo(abs_filter_counter + 1,10);
abs_filter(1,11) = abs_pseudo(abs_filter_counter + 1,11);
abs_filter(1,12) = abs_pseudo(abs_filter_counter + 1,12);
```

```
abs_filter(1,13) = abs_pseudo(abs_filter_counter + 1,13);
abs_filter(1,14) = abs_pseudo(abs_filter_counter + 1,14);

% collect the duration of days
% of the simultaneous absorption events.
% note: all the values for the duration of days
% are collected except the last one.
% the last value is added outside the while loop
abs_filter_duration_days(abs_index) = ...
abs_pseudo(abs_filter_counter, 15);

% collect the duration of hours
% of the simultaneous absorption events.
% note: all the values for the duration of hours
% are collected except the last one.
% the last value is added outside the while loop.
abs_filter_duration_hours(abs_index) = ...
abs_pseudo(abs_filter_counter, 16);

% collect the duration of minutes
% of the simultaneous absorption events.
% note: all the values for the duration of minutes
% are collected except the last one.
% the last value is added outside the while loop.
abs_filter_duration_mins(abs_index) = ...
abs_pseudo(abs_filter_counter, 17);

% collect the duration of seconds
% of the simultaneous absorption events.
% note: all the values for the duration of secondss
% are collected except the last one.
% the last value is added outside the while loop.
abs_filter_duration_secs(abs_index) = ...
abs_pseudo(abs_filter_counter, 18);

% collect the max values of the simultaneous absorpton
events.
```



```
% note: all the max values are collected except the last
one.

% the last value is added outside the while loop,
% one line before the abs_filter(1, 19) variable is
called.

abs_filter_max(abs_index) = ...
abs_pseudo(abs_filter_counter, 19);

% similarly collect their mean values.
% note: all the mean values are collected except the last
one.

% the last value is added outside the while loop,
% one line before the abs_filter(1, 20) variable is
called.

abs_filter_mean(abs_index) = ...
abs_pseudo(abs_filter_counter, 20);

% increament the counters by one.
abs_filter_counter = abs_filter_counter + 1;
abs_index = abs_index + 1;

end % end of the while loop used to find the number of the
    % simultaneous pseudo absorption events.

% find the duration of days.
% add the last values for the duration of days as well.
abs_filter_duration_days = ...
[abs_filter_duration_days ...
abs_pseudo(abs_filter_counter, 15)];
abs_filter_duration_sum_days =
sum(abs_filter_duration_days);
abs_filter_duration_tot_days = ...
timespan(abs_filter_duration_sum_days , 'd');

% this is not used anywhere, but may be needed later.
```

```
abs_filter(1,15) = abs_filter_duration_sum_days;

% find the duration of hours.
% add the last values for the duration of hours as
well.

abs_filter_duration_hours = ...
[abs_filter_duration_hours ...
abs_pseudo(abs_filter_counter, 16)];
abs_filter_duration_sum_hours =
sum(abs_filter_duration_hours);
abs_filter_duration_tot_hours = ...
timespan(abs_filter_duration_sum_hours , 'h');

% this is not used anywhere, but may be needed later.
abs_filter(1,16) = abs_filter_duration_sum_hours;

% find the duration of minutes.
% add the last values for the duration of minutes as
well.

abs_filter_duration_mins = ...
[abs_filter_duration_mins ...
abs_pseudo(abs_filter_counter, 17)];
abs_filter_duration_sum_mins =
sum(abs_filter_duration_mins);
abs_filter_duration_tot_mins = ...
timespan(abs_filter_duration_sum_mins , 'm');

% this is not used anywhere, but may be needed later.
abs_filter(1,17) = abs_filter_duration_sum_mins;

% find the duration of seconds.
% add the last values for the duration of seconds as
well.

abs_filter_duration_secs = ...
[abs_filter_duration_secs ...
abs_pseudo(abs_filter_counter, 18)];
abs_filter_duration_sum_secs =
```

```
sum(abs_filter_duration_secs);
    abs_filter_duration_tot_secs = ...
timespan(abs_filter_duration_sum_secs , 's');

% this is not used anywhere, but may be needed later.
abs_filter(1,18) = abs_filter_duration_sum_secs;

% find the total duration.
abs_filter_duration_total = ...
abs_filter_duration_tot_days + ...
abs_filter_duration_tot_hours + ...
abs_filter_duration_tot_mins + ...
abs_filter_duration_tot_secs;

% add to the abs_filter_max matrix the max value of
% the last absorption event.
abs_filter_max = ...
[abs_filter_max abs_pseudo(abs_filter_counter, 19)];

% find the max value.
abs_filter(1,19) = max(abs_filter_max);

% add to the abs_filter_mean matrix the mean value of
% the last absorption event.
abs_filter_mean = ...
[abs_filter_mean abs_pseudo(abs_filter_counter, 20)];

% find the duration of each pseudo absorption event.
for abs_duration_index =
1:length(abs_filter_duration_days)

    % find the duration of days of a pseudo absorption
event.
    abs_filter_mean_duration_days(abs_duration_index) = ...
timespan( ...
abs_filter_duration_days(abs_duration_index) , 'd');
    % find the duration of hours of a pseudo absorption
```

```

event.
    abs_filter_mean_duration_hours(abs_duration_index) = ...
timespan( ...
abs_filter_duration_hours(abs_duration_index) , 'h');
    % find the duration of minutes of a pseudo absorption
event.
    abs_filter_mean_duration_mins(abs_duration_index) = ...
timespan( ...
abs_filter_duration_mins(abs_duration_index) , 'm');
    % find the duration of seconds of a pseudo absorption
event.
    abs_filter_mean_duration_secs(abs_duration_index) = ...
timespan( ...
abs_filter_duration_secs(abs_duration_index) , 's');

    % find the total duration of a pseudo absorption event.
    abs_filter_mean_duration(abs_duration_index) = ...
abs_filter_mean_duration_days(abs_duration_index) + ...
abs_filter_mean_duration_hours(abs_duration_index) + ...
abs_filter_mean_duration_mins(abs_duration_index) + ...
abs_filter_mean_duration_secs(abs_duration_index);

end % end of for loop for the duration
    % of each pseudo absorption event.

% find the correct mean value
% of each pseudo absorption event.
for abs_mean_index = 1:length(abs_filter_duration_days)

    abs_filter_mean_correct(abs_mean_index) = ...
(abs_filter_mean_duration(abs_mean_index)/...
abs_filter_duration_total) * ...
abs_filter_mean(abs_mean_index);

end % end of for loop for the mean value
    % of the filtered absorption event.

```

```
% find the correct mean value
% of the filtered absorption event.
abs_filter(1, 20) = ...
sum(abs_filter_mean_correct);

% find the type of absorption event.
%
% return '4' for Polar Cap Absorption (PCA).
% return '3' for Auroral Absorption (AA).
% return '2' for Data Gap.
% return '1' for other (general) Absorption.

% characteristics of Polar Cap Absorption.
pca_max_duration = timespan(10, 'd');
pca_min_duration = timespan(8, 'h');
pca_min_value = 1;
pca_max_value = 100;

% characteristics of Auroral Absorption.
aa_max_duration = timespan(4, 'h');
aa_min1_duration = timespan(2, 'm');
aa_min2_duration = timespan(6, 'm');
aa_min1_value = 0.5;
aa_min2_value = 1;
aa_max_value = 100;

% if the statement below is true, then PCA has occurred.
if (abs_filter_duration_total >= pca_min_duration & ...
    abs_filter_duration_total <= pca_max_duration) & ...
    (abs_filter(1,19) >= pca_min_value & ...
    abs_filter(1,19) <= pca_max_value) & ...
    ~isnan(abs_filter(1, 19))

% represent the Polar Cap Absorption with '5'.
abs_event = 5;
```

```

end % end of if statement for PCA.

% if the statement below is true, then AA has occurred.
if ((abs_filter_duration_total >= aa_min2_duration & ...
    abs_filter_duration_total <= aa_max_duration) & ...
    (abs_filter(1,19) >= aa_min1_value & ...
    abs_filter(1,19) <= aa_max_value) & ...
    ~isnan(abs_filter(1, 19))) | ...
    ((abs_filter_duration_total >= aa_min1_duration & ...
    abs_filter_duration_total <= aa_min2_duration) & ...
    (abs_filter(1,19) >= aa_min2_value & ...
    abs_filter(1,19) <= aa_max_value) & ...
    ~isnan(abs_filter(1, 19)))

% represent the Auroral Absorption with '4'.
abs_event = 4;

end % end of if statement for AA.

% return the results of the filtered absorption events.
fprintf(fid_filter, ...
    '%d %d %d %d %d %d %d %d %d %d %d %d %d %d %s %s %s %s %.2f'
    '%.2f', ...
    abs_event, ...
    abs_filter(1, 2), ...
    abs_filter(1, 3), ...
    abs_filter(1, 4), ...
    abs_filter(1, 5), ...
    abs_filter(1, 6), ...
    abs_filter(1, 7), ...
    abs_filter(1, 8), ...
    abs_filter(1, 9), ...
    abs_filter(1,10), ...
    abs_filter(1,11), ...
    abs_filter(1,12), ...

```

```
abs_filter(1,13), ...
abs_filter(1,14), ...
strftime(abs_filter_duration_total, '%d'), ...
    strftime(abs_filter_duration_total, '%H'), ...
    strftime(abs_filter_duration_total, '%M'), ...
    strftime(abs_filter_duration_total, '%S'), ...
abs_filter(1,19), ...
abs_filter(1,20));
fprintf(fid_filter, '\n');

% initialise the various variables.
abs_filter_max = [];
abs_filter_mean = [];
abs_filter_duration_days = [];
abs_filter_duration_hours = [];
abs_filter_duration_mins = [];
abs_filter_duration_secs = [];
abs_filter_mean_duration = timespan(0, 's');
abs_filter_mean_correct = 0;

% reinitialise the matrix where the results of the
% filter analysis will be stored each time it loops
% the while loop below.
abs_filter = zeros(1,abs_pseudo_size(2));

% reinitialise the counter for the pseudo absorption
% events to the next pseudo absorption occurrence.
abs_pseudo_occurrence = abs_filter_counter;

% if the conditions below are true
% then data gap event has occurred.
%
% condition 1: it should be a data gap event (i.e. '2'),
%             1 for absorption, 2 for data gap.
% condition 2: the end time of a specific data gap event,
%             must be date(Y M d) time(h 00 s).
```

```
% condition 3: the start time of the next data gap event,
%               must be date(Y M d) time(h 00 s).
% condition 4: check that the data gap events have
%               occurred on the same beam.
elseif abs_pseudo(abs_filter_counter, 1) == 2 ...
& abs_pseudo(abs_filter_counter, 13) == 0 ...
& abs_pseudo(abs_filter_counter + 1, 7) == 0 ...
& (abs_pseudo(abs_filter_counter, 1) == ...
abs_pseudo(abs_filter_counter + 1, 1)) ...
& (abs_pseudo(abs_filter_counter, 2) == ...
abs_pseudo(abs_filter_counter + 1, 2))

% initialise a variable, it will be used
% for indexing in the while loop below.
abs_index = 1;

% use a while loop to find the number of pseudo
% data gap events have occurred simultaneously.
while abs_pseudo(abs_filter_counter, 13) == 0 ...
& abs_pseudo(abs_filter_counter + 1, 7) == 0 ...
& (abs_pseudo(abs_filter_counter, 1) == ...
abs_pseudo(abs_filter_counter + 1, 1)) ...
& (abs_pseudo(abs_filter_counter, 2) == ...
abs_pseudo(abs_filter_counter + 1, 2))

% find the event.
% this is still the non-analysed data gap event,
% but is used for initialising the variable.
abs_filter(1, 1) = abs_pseudo(abs_filter_counter, 1);

% find the beam.
abs_filter(1, 2) = abs_pseudo(abs_filter_counter, 2);

% find the start time.
abs_filter(1, 3) = abs_pseudo(abs_pseudo_occurrence, 3);
abs_filter(1, 4) = abs_pseudo(abs_pseudo_occurrence, 4);
abs_filter(1, 5) = abs_pseudo(abs_pseudo_occurrence, 5);
```



```
abs_filter(1, 6) = abs_pseudo(abs_pseudo_occurrence, 6);
abs_filter(1, 7) = abs_pseudo(abs_pseudo_occurrence, 7);
abs_filter(1, 8) = abs_pseudo(abs_pseudo_occurrence, 8);

% find the end time.
abs_filter(1, 9) = abs_pseudo(abs_filter_counter + 1, 9);
abs_filter(1,10) = abs_pseudo(abs_filter_counter + 1,10);
abs_filter(1,11) = abs_pseudo(abs_filter_counter + 1,11);
abs_filter(1,12) = abs_pseudo(abs_filter_counter + 1,12);
abs_filter(1,13) = abs_pseudo(abs_filter_counter + 1,13);
abs_filter(1,14) = abs_pseudo(abs_filter_counter + 1,14);

% collect the duration of days
% of the simultaneous data gap events.
% note: all the values for the duration of days
% are collected except the last one.
% the last value is added outside the while loop.
abs_filter_duration_days(abs_index) = ...
abs_pseudo(abs_filter_counter, 15);

% collect the duration of hours
% of the simultaneous data gap events.
% note: all the values for the duration of hours
% are collected except the last one.
% the last value is added outside the while loop.
abs_filter_duration_hours(abs_index) = ...
abs_pseudo(abs_filter_counter, 16);

% collect the duration of minutes
% of the simultaneous data gap events.
% note: all the values for the duration of minutes
% are collected except the last one.
% the last value is added outside the while loop.
abs_filter_duration_mins(abs_index) = ...
abs_pseudo(abs_filter_counter, 17);

% collect the duration of seconds
```

```
% of the simultaneous data gap events.
% note: all the values for the duration of seconds
% are collected except the last one.
% the last value is added outside the while loop.
abs_filter_duration_secs(abs_index) = ...
abs_pseudo(abs_filter_counter, 18);

% collect the max values of the simultaneous data gap
events.
abs_filter_max(abs_index) = ...
abs_pseudo(abs_filter_counter, 19);

% find the mean value.
abs_filter_mean(abs_index) = ...
abs_pseudo(abs_filter_counter, 20);

% increament the counters by one.
abs_filter_counter = abs_filter_counter + 1;
abs_index = abs_index + 1;

end % end of the while loop used to find the number of the
    % simultaneous pseudo data gap events.

% find the duration of days.
% add the last values for the duration of days as well.
abs_filter_duration_days = ...
[abs_filter_duration_days ...
abs_pseudo(abs_filter_counter, 15)];
abs_filter_duration_sum_days =
sum(abs_filter_duration_days);
abs_filter_duration_tot_days = ...
timespan(abs_filter_duration_sum_days , 'd');

% this is not used anywhere, but may be needed later.
abs_filter(1,15) = abs_filter_duration_sum_days;

% find the duration of hours.
```

```
% add the last values for the duration of hours as well.
abs_filter_duration_hours = ...
[abs_filter_duration_hours ...
abs_pseudo(abs_filter_counter, 16)];
abs_filter_duration_sum_hours =
sum(abs_filter_duration_hours);
abs_filter_duration_tot_hours = ...
timespan(abs_filter_duration_sum_hours , 'h');

% this is not used anywhere, but may be needed later.
abs_filter(1,16) = abs_filter_duration_sum_hours;

% find the duration of minutes.
% add the last values for the duration of minutes as
well.
abs_filter_duration_mins = ...
[abs_filter_duration_mins ...
abs_pseudo(abs_filter_counter, 17)];
abs_filter_duration_sum_mins =
sum(abs_filter_duration_mins);
abs_filter_duration_tot_mins = ...
timespan(abs_filter_duration_sum_mins , 'm');

% this is not used anywhere, but may be needed later.
abs_filter(1,17) = abs_filter_duration_sum_mins;

% find the duration of seconds.
% add the last values for the duration of seconds as
well.
abs_filter_duration_secs = ...
[abs_filter_duration_secs ...
abs_pseudo(abs_filter_counter, 18)];
abs_filter_duration_sum_secs =
sum(abs_filter_duration_secs);
abs_filter_duration_tot_secs = ...
timespan(abs_filter_duration_sum_secs , 's');
```

```
% this is not used anywhere, but may be needed later.
abs_filter(1,18) = abs_filter_duration_sum_secs;

% find the total duration.
abs_filter_duration_total = ...
abs_filter_duration_tot_days + ...
abs_filter_duration_tot_hours + ...
abs_filter_duration_tot_mins + ...
abs_filter_duration_tot_secs;

% find the max value.
abs_filter(1,19) = max(abs_filter_max);

% find the mean value.
abs_filter(1,20) = mean(abs_filter_mean);

% find the data gap events.
%
% return '2' for Data Gap.

% the characteristics of Data Gap have been
% stated in irisdata_pseudoanalysis function.

% if the statement below is true, then Data Gap has
occured.
if (abs_filter(1, 1) == 2)

% represent the Data Gap with '2'.
abs_event = 2;

end % end of if statement for Data Gap.

% return the results of the filtered absorption events.
fprintf(fid_filter, ...
'%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %s %s %s %s %.2f
```

```
%.2f', ...
    abs_event, ...
    abs_filter(1, 2), ...
    abs_filter(1, 3), ...
    abs_filter(1, 4), ...
    abs_filter(1, 5), ...
    abs_filter(1, 6), ...
    abs_filter(1, 7), ...
    abs_filter(1, 8), ...
    abs_filter(1, 9), ...
    abs_filter(1,10), ...
    abs_filter(1,11), ...
    abs_filter(1,12), ...
    abs_filter(1,13), ...
    abs_filter(1,14), ...
    strftime(abs_filter_duration_total, '%d'), ...
        strftime(abs_filter_duration_total, '%H'), ...
        strftime(abs_filter_duration_total, '%M'), ...
        strftime(abs_filter_duration_total, '%S'), ...
    abs_filter(1,19), ...
    abs_filter(1,20));
fprintf(fid_filter, '\n');

% initialise the various variables.
abs_filter_max = [];
abs_filter_mean = [];
abs_filter_duration_days = [];
abs_filter_duration_hours = [];
abs_filter_duration_mins = [];
abs_filter_duration_secs = [];

% reinitialise the matrix where the results of the
% filter analysis will be stored each time it loops
% the while loop below.
abs_filter = zeros(1,abs_pseudo_size(2));

% reinitialise the counter for the pseudo absorption
```

```
% events to the next pseudo absorption occurrence.
abs_pseudo_occurrence = abs_filter_counter;

% if the conditions below are true
% then pseudo noise event has occurred.
%
% condition 1: it should be an absorption event (i.e.
'3'),
%           1 for absorption, 2 for data gap.
% condition 2: the end time of a specific absorption
event,
%           must be date(Y M d) time(h 00 s).
% condition 3: the start time of the next absorption
event,
%           must be date(Y M d) time(h 00 s).
% condition 4: check that the absorption events have
%           occurred on the same beam.
elseif abs_pseudo(abs_filter_counter, 1) == 3 ...
& abs_pseudo(abs_filter_counter, 13) == 0 ...
& abs_pseudo(abs_filter_counter + 1, 7) == 0 ...
& (abs_pseudo(abs_filter_counter, 1) == ...
abs_pseudo(abs_filter_counter + 1, 1)) ...
& (abs_pseudo(abs_filter_counter, 2) == ...
abs_pseudo(abs_filter_counter + 1, 2))

% initialise a variable, it will be used
% for indexing in the while loop below.
abs_index = 1;

% use a while loop to find the number of pseudo
% absorption events have occurred simultaneously.
while abs_pseudo(abs_filter_counter, 13) == 0 ...
& abs_pseudo(abs_filter_counter + 1, 7) == 0 ...
& (abs_pseudo(abs_filter_counter, 1) == ...
abs_pseudo(abs_filter_counter + 1, 1)) ...
& (abs_pseudo(abs_filter_counter, 2) == ...
```

```
abs_pseudo(abs_filter_counter + 1, 2))

% find the event.
% this is still the non-analysed noise event,
% but is used for initialising the variable.
abs_filter(1, 1) = abs_pseudo(abs_filter_counter, 1);

% find the beam.
abs_filter(1, 2) = abs_pseudo(abs_filter_counter, 2);

% find the start time.
abs_filter(1, 3) = abs_pseudo(abs_pseudo_occurrence, 3);
abs_filter(1, 4) = abs_pseudo(abs_pseudo_occurrence, 4);
abs_filter(1, 5) = abs_pseudo(abs_pseudo_occurrence, 5);
abs_filter(1, 6) = abs_pseudo(abs_pseudo_occurrence, 6);
abs_filter(1, 7) = abs_pseudo(abs_pseudo_occurrence, 7);
abs_filter(1, 8) = abs_pseudo(abs_pseudo_occurrence, 8);

% find the end time.
abs_filter(1, 9) = abs_pseudo(abs_filter_counter + 1,
9);
abs_filter(1,10) = abs_pseudo(abs_filter_counter +
1,10);
abs_filter(1,11) = abs_pseudo(abs_filter_counter +
1,11);
abs_filter(1,12) = abs_pseudo(abs_filter_counter +
1,12);
abs_filter(1,13) = abs_pseudo(abs_filter_counter +
1,13);
abs_filter(1,14) = abs_pseudo(abs_filter_counter +
1,14);

% collect the duration of days
% of the simultaneous noise events.
% note: all the values for the duration of days
% are collected except the last one.
% the last value is added outside the while loop
```

```
abs_filter_duration_days(abs_index) = ...
abs_pseudo(abs_filter_counter, 15);

% collect the duration of hours
% of the simultaneous noise events.
% note: all the values for the duration of hours
% are collected except the last one.
% the last value is added outside the while loop.
abs_filter_duration_hours(abs_index) = ...
abs_pseudo(abs_filter_counter, 16);

% collect the duration of minutes
% of the simultaneous noise events.
% note: all the values for the duration of minutes
% are collected except the last one.
% the last value is added outside the while loop.
abs_filter_duration_mins(abs_index) = ...
abs_pseudo(abs_filter_counter, 17);

% collect the duration of seconds
% of the simultaneous noise events.
% note: all the values for the duration of seconds
% are collected except the last one.
% the last value is added outside the while loop.
abs_filter_duration_secs(abs_index) = ...
abs_pseudo(abs_filter_counter, 18);

% collect the max values of the simultaneous noise
events.
% note: all the max values are collected except the last
one.
% the last value is added outside the while loop,
% one line before the abs_filter(1, 19) variable is
called.
abs_filter_max(abs_index) = ...
abs_pseudo(abs_filter_counter, 19);
```



```
% similarly collect their mean values.
% note: all the mean values are collected except the
last one.
% the last value is added outside the while loop,
% one line before the abs_filter(1, 20) variable is
called.
abs_filter_mean(abs_index) = ...
abs_pseudo(abs_filter_counter, 20);

% increament the counters by one.
abs_filter_counter = abs_filter_counter + 1;
abs_index = abs_index + 1;

end % end of the while loop used to find the number of the
    % simultaneous pseudo noise events.

% find the duration of days.
% add the last values for the duration of days as well.
abs_filter_duration_days = ...
[abs_filter_duration_days ...
abs_pseudo(abs_filter_counter, 15)];
abs_filter_duration_sum_days =
sum(abs_filter_duration_days);
abs_filter_duration_tot_days = ...
timespan(abs_filter_duration_sum_days , 'd');

% this is not used anywhere, but may be needed later.
abs_filter(1,15) = abs_filter_duration_sum_days;

% find the duration of hours.
% add the last values for the duration of hours as
well.
abs_filter_duration_hours = ...
[abs_filter_duration_hours ...
abs_pseudo(abs_filter_counter, 16)];
```

```
abs_filter_duration_sum_hours =  
sum(abs_filter_duration_hours);  
abs_filter_duration_tot_hours = ...  
timespan(abs_filter_duration_sum_hours , 'h');  
  
% this is not used anywhere, but may be needed later.  
abs_filter(1,16) = abs_filter_duration_sum_hours;  
  
% find the duration of minutes.  
% add the last values for the duration of minutes as  
well.  
abs_filter_duration_mins = ...  
[abs_filter_duration_mins ...  
abs_pseudo(abs_filter_counter, 17)];  
abs_filter_duration_sum_mins =  
sum(abs_filter_duration_mins);  
abs_filter_duration_tot_mins = ...  
timespan(abs_filter_duration_sum_mins , 'm');  
  
% this is not used anywhere, but may be needed later.  
abs_filter(1,17) = abs_filter_duration_sum_mins;  
  
% find the duration of seconds.  
% add the last values for the duration of seconds as  
well.  
abs_filter_duration_secs = ...  
[abs_filter_duration_secs ...  
abs_pseudo(abs_filter_counter, 18)];  
abs_filter_duration_sum_secs =  
sum(abs_filter_duration_secs);  
abs_filter_duration_tot_secs = ...  
timespan(abs_filter_duration_sum_secs , 's');  
  
% this is not used anywhere, but may be needed later.  
abs_filter(1,18) = abs_filter_duration_sum_secs;  
  
% find the total duration.
```

```
abs_filter_duration_total = ...
abs_filter_duration_tot_days + ...
abs_filter_duration_tot_hours + ...
abs_filter_duration_tot_mins + ...
abs_filter_duration_tot_secs;

% add to the abs_filter_max matrix the max value of
% the last noise event.
abs_filter_max = ...
[abs_filter_max abs_pseudo(abs_filter_counter, 19)];

% find the min value.
% (because the noise is negative we find the minimum).
abs_filter(1,19) = min(abs_filter_max);

% add to the abs_filter_mean matrix the mean value of
% the last noise event.
abs_filter_mean = ...
[abs_filter_mean abs_pseudo(abs_filter_counter, 20)];

% find the duration of each pseudo noise event.
for abs_duration_index =
1:length(abs_filter_duration_days)

    % find the duration of days of a pseudo noise event.
    abs_filter_mean_duration_days(abs_duration_index) = ...
timespan( ...
abs_filter_duration_days(abs_duration_index) , 'd');
    % find the duration of hours of a pseudo noise event.
    abs_filter_mean_duration_hours(abs_duration_index) = ...
timespan( ...
abs_filter_duration_hours(abs_duration_index) , 'h');
    % find the duration of minutes of a pseudo noise event.
    abs_filter_mean_duration_mins(abs_duration_index) = ...
timespan( ...
abs_filter_duration_mins(abs_duration_index) , 'm');
    % find the duration of seconds of a pseudo noise event.
```

```
abs_filter_mean_duration_secs(abs_duration_index) = ...
timespan( ...
abs_filter_duration_secs(abs_duration_index) , 's');

% find the total duration of a pseudo noise event.
abs_filter_mean_duration(abs_duration_index) = ...
abs_filter_mean_duration_days(abs_duration_index) + ...
abs_filter_mean_duration_hours(abs_duration_index) + ...
abs_filter_mean_duration_mins(abs_duration_index) + ...
abs_filter_mean_duration_secs(abs_duration_index);

end % end of for loop for the duration
    % of each pseudo noise event.

% find the correct mean value
% of each pseudo noise event.
for abs_mean_index = 1:length(abs_filter_duration_days)

    abs_filter_mean_correct(abs_mean_index) = ...
(abs_filter_mean_duration(abs_mean_index)/...
abs_filter_duration_total) * ...
abs_filter_mean(abs_mean_index);

end % end of for loop for the mean value
    % of the filtered noise event.

% find the correct mean value
% of the filtered absorption event.
abs_filter(1, 20) = ...
sum(abs_filter_mean_correct);

% find the type of noise event.
%
% return '5' for Polar Cap Absorption (PCA).
% return '4' for Auroral Absorption (AA).
```

```
% return '3' for Noise.
% return '2' for Data Gap.
% return '1' for other (general) Absorption.

% characteristics of noise events,
noise_max_duration = timespan(100,'d');
noise_min_duration = timespan(1,'m');
noise_min_value = -1;
noise_max_value = -100;

% if the statement below is true, then Noise has
occured.
if (abs_filter_duration_total >= noise_min_duration &
...
abs_filter_duration_total <= noise_max_duration) & ...
(abs_filter(1,19) <= noise_min_value & ...
abs_filter(1,19) >= noise_max_value) & ...
~isnan(abs_filter(1, 19))

% represent the Noise with '3'.
abs_event = 3;

end % end of if statement for Noise.

% return the results of the filtered absorption events.
fprintf(fid_filter, ...
'%d %d %d %d %d %d %d %d %d %d %d %d %d %d %s %s %s %s %.2f
%.2f', ...
abs_event, ...
abs_filter(1, 2), ...
abs_filter(1, 3), ...
abs_filter(1, 4), ...
abs_filter(1, 5), ...
abs_filter(1, 6), ...
abs_filter(1, 7), ...
abs_filter(1, 8), ...
abs_filter(1, 9), ...
```

```
abs_filter(1,10), ...
abs_filter(1,11), ...
abs_filter(1,12), ...
abs_filter(1,13), ...
abs_filter(1,14), ...
strftime(abs_filter_duration_total, '%d'), ...
    strftime(abs_filter_duration_total, '%H'), ...
    strftime(abs_filter_duration_total, '%M'), ...
    strftime(abs_filter_duration_total, '%S'), ...
abs_filter(1,19), ...
abs_filter(1,20));
fprintf(fid_filter, '\n');

% initialise the various variables.
abs_filter_max = [];
abs_filter_mean = [];
abs_filter_duration_days = [];
abs_filter_duration_hours = [];
abs_filter_duration_mins = [];
abs_filter_duration_secs = [];
abs_filter_mean_duration = timespan(0, 's');
abs_filter_mean_correct = 0;

% reinitialise the matrix where the results of the
% filter analysis will be stored each time it loops
% the while loop below.
abs_filter = zeros(1,abs_pseudo_size(2));

% reinitialise the counter for the pseudo noise
% events to the next pseudo noise occurrence.
abs_pseudo_occurrence = abs_filter_counter;

% make the analysis on the events that have not
% been splitted between {h:59:s} and {h+1:00:s}.
else
```

```
% assign the same values of the current event
% into abs_filter matrix.
abs_filter(1, 1) = abs_pseudo(abs_filter_counter, 1);
abs_filter(1, 2) = abs_pseudo(abs_filter_counter, 2);
abs_filter(1, 3) = abs_pseudo(abs_filter_counter, 3);
abs_filter(1, 4) = abs_pseudo(abs_filter_counter, 4);
abs_filter(1, 5) = abs_pseudo(abs_filter_counter, 5);
abs_filter(1, 6) = abs_pseudo(abs_filter_counter, 6);
abs_filter(1, 7) = abs_pseudo(abs_filter_counter, 7);
abs_filter(1, 8) = abs_pseudo(abs_filter_counter, 8);
abs_filter(1, 9) = abs_pseudo(abs_filter_counter, 9);
abs_filter(1, 10) = abs_pseudo(abs_filter_counter, 10);
abs_filter(1, 11) = abs_pseudo(abs_filter_counter, 11);
abs_filter(1, 12) = abs_pseudo(abs_filter_counter, 12);
abs_filter(1, 13) = abs_pseudo(abs_filter_counter, 13);
abs_filter(1, 14) = abs_pseudo(abs_filter_counter, 14);
abs_filter(1, 15) = abs_pseudo(abs_filter_counter, 15);
abs_filter(1, 16) = abs_pseudo(abs_filter_counter, 16);
abs_filter(1, 17) = abs_pseudo(abs_filter_counter, 17);
abs_filter(1, 18) = abs_pseudo(abs_filter_counter, 18);
abs_filter(1, 19) = abs_pseudo(abs_filter_counter, 19);
abs_filter(1, 20) = abs_pseudo(abs_filter_counter, 20);

% find the duration of days.
abs_filter_duration_tot_days = ...
timespan(abs_filter(1, 15) , 'd');

% find the duration of hours.
abs_filter_duration_tot_hours = ...
timespan(abs_filter(1, 16) , 'h');

% find the duration of minutes.
abs_filter_duration_tot_mins = ...
timespan(abs_filter(1, 17) , 'm');

% find the duration of seconds.
abs_filter_duration_tot_secs = ...
```

```
timespan(abs_filter(1, 18) , 's');

% find the total duration.
abs_filter_duration_total = ...
abs_filter_duration_tot_days + ...
abs_filter_duration_tot_hours + ...
abs_filter_duration_tot_mins + ...
abs_filter_duration_tot_secs;

% find the type of absorption event.
%
% return '4' for Polar Cap Absorption (PCA).
% return '3' for Auroral Absorption (AA).
% return '2' for Data Gap.
% return '1' for other (general) Absorption.

% characteristics of Polar Cap Absorption.
pca_max_duration = timespan(10, 'd');
pca_min_duration = timespan(8, 'h');
pca_min_value = 1;
pca_max_value = 100;

% characteristics of Auroral Absorption.
aa_max_duration = timespan(4, 'h');
aa_min1_duration = timespan(2, 'm');
aa_min2_duration = timespan(6, 'm');
aa_min1_value = 0.5;
aa_min2_value = 1;
aa_max_value = 100;

% the characteristics of Data Gap have been
% stated in irisdata_pseudoanalysis function.

% characteristics of noise events,
noise_max_duration = timespan(100, 'd');
noise_min_duration = timespan(1, 'm');
```



```
noise_min_value = -1;
noise_max_value = -100;

% if the statement below is true, then PCA has occurred.
if (abs_filter_duration_total >= pca_min_duration & ...
    abs_filter_duration_total <= pca_max_duration) & ...
    (abs_filter(1,19) >= pca_min_value & ...
    abs_filter(1,19) <= pca_max_value) & ...
    ~isnan(abs_filter(1, 19))

% represent the Polar Cap Absorption with '5'.
abs_filter(1, 1) = 5;

end % end of if statement for PCA.

% if the statement below is true, then AA has occurred.
if ((abs_filter_duration_total >= aa_min2_duration & ...
    abs_filter_duration_total <= aa_max_duration) & ...
    (abs_filter(1,19) >= aa_min1_value & ...
    abs_filter(1,19) <= aa_max_value) & ...
    ~isnan(abs_filter(1, 19))) | ...
    ((abs_filter_duration_total >= aa_min1_duration & ...
    abs_filter_duration_total <= aa_min2_duration) & ...
    (abs_filter(1,19) >= aa_min2_value & ...
    abs_filter(1,19) <= aa_max_value) & ...
    ~isnan(abs_filter(1, 19)))

% represent the Auroral Absorption with '4'.
abs_filter(1, 1) = 4;

end % end of if statement for AA.

% if the statement below is true, then Noise has
occured.
if (abs_filter_duration_total >= noise_min_duration &
...
    abs_filter_duration_total <= noise_max_duration) & ...
```

```

        (abs_filter(1,19) <= noise_min_value & ...
        abs_filter(1,19) >= noise_max_value) & ...
        ~isnan(abs_filter(1, 19))

% represent the Noise with '3'.
abs_filter(1, 1) = 3;

end % end of if statement for Noise.

% if the statement below is true, then Data Gap has
occured.
if (abs_filter(1, 1) == 2 & isnan(abs_filter(1, 19)))

% represent the Data Gap with '2'.
abs_filter(1, 1) = 2;

end % end of if statement for Data Gap.

% return the results of the non filtered events.
fprintf(fid_filter, ...
'%d %d %d %d %d %d %d %d %d %d %d %d %d %d %s %s %s %s %.2f
%.2f', ...
abs_filter(1, 1), ...
abs_filter(1, 2), ...
abs_filter(1, 3), ...
abs_filter(1, 4), ...
abs_filter(1, 5), ...
abs_filter(1, 6), ...
abs_filter(1, 7), ...
abs_filter(1, 8), ...
abs_filter(1, 9), ...
abs_filter(1, 10), ...
abs_filter(1, 11), ...
abs_filter(1, 12), ...
abs_filter(1, 13), ...
abs_filter(1, 14), ...

```

```
    strftime(abs_filter_duration_total, '%d'), ...
    strftime(abs_filter_duration_total, '%H'), ...
    strftime(abs_filter_duration_total, '%M'), ...
    strftime(abs_filter_duration_total, '%S'), ...
    abs_filter(1, 19), ...
    abs_filter(1, 20));
fprintf(fid_filter, '\n');

% initialise the various variables.
abs_filter_duration_days = [];
abs_filter_duration_hours = [];
abs_filter_duration_mins = [];
abs_filter_duration_secs = [];

% reinitialise the counter for the pseudo absorption
% events to the next pseudo absorption occurrence.
abs_pseudo_occurrence = abs_filter_counter;

end % end of the if-elseif-else statement for the
    % filter absorption, noise and data gap events.

% reinitialise the matrix where the results of the
% filter analysis will be stored each time it loops
% the while loop below.
abs_filter = zeros(1,abs_pseudo_size(2));

% increament the counter for the pseudo absorption
% events by one.
abs_pseudo_occurrence = abs_pseudo_occurrence + 1;

end % end of the while loop,
    % used for the pseudo absorption occurrences.

% close the file that has been opened to save the results.
status = fclose(fid_filter);
```

irisdata_load.m

```
function [data_values, current_beam, current_starttime, ...
        current_endtime, current_duration, current_datasize] ...
    = irisdata_load(data_currenttime, data_duration, ...
        data_resolution, beams_requested);

%IRISDATA_LOAD
%
%Date Started: 26/05/2000 5:48pm
%Date Completed: 27/05/2000 3:39am
%Creator: G.Nikitas
%DCS, Lancaster University
%
% once the appropriate inputs have been provided by the user
% the function loads the correct data, depending on the
% user's requests.

% create the requested data.
iabs = irisabs(data_currenttime, data_duration, ...
    data_resolution, kil, beams_requested);

% get the contents (values) of the data and store them in a
matrix.
data_values = getabs(iabs);

% collect useful information about the data.
current_beam = getbeams(iabs);
current_starttime = getstarttime(iabs);
current_endtime = getendtime(iabs);
current_duration = getduration(iabs);
current_datasize = size(data_values);
```

createcatalogue.m

```
function createcatalogue(year, months);

%CREATECATALOGUE
%
%Date Started: 18/06/2000 9:57pm
%Date Completed:
%Creator: G.Nikitas
%DCS, Lancaster University
%
% run this script to concatenate the files
% with results of the analysis for all the months
% and for the current year.
% note that if the fuction is called more than once
% for the same year, the contents of the results file
% (e.g. zcatalogue1995.gn) will be overwritten.

% set the default value for the months.
if nargin < 2
    months = [1:12];
end

% initialise the results variables.
r1 = [];
r2 = [];
r3 = [];
r4 = [];
r5 = [];
r6 = [];
r7 = [];
r8 = [];
r9 = [];
r10 = [];
r11 = [];
r12 = [];
```

```
% find the start time.
st = time([year 01 01 0 0 0]);

% find the next year. ie the end time.
next_year = year + 1;
ed = time([next_year 01 01 0 0 0]);

% go through the months.
for currentmonth = months

    if currentmonth == 1
        r1 = load(['zfilter' strftime(st, '%Y') '0101_' ...
                  strftime(st, '%Y') '0201.gn']);
    end % end of if statement of the 1st month.

    if currentmonth == 2
        r2 = load(['zfilter' strftime(st, '%Y') '0201_' ...
                  strftime(st, '%Y') '0301.gn']);
    end % end of if statement of the 2nd month.

    if currentmonth == 3
        r3 = load(['zfilter' strftime(st, '%Y') '0301_' ...
                  strftime(st, '%Y') '0401.gn']);
    end % end of if statement of the 3rd month.

    if currentmonth == 4
        r4 = load(['zfilter' strftime(st, '%Y') '0401_' ...
                  strftime(st, '%Y') '0501.gn']);
    end % end of if statement of the 4th month.

    if currentmonth == 5
        r5 = load(['zfilter' strftime(st, '%Y') '0501_' ...
                  strftime(st, '%Y') '0601.gn']);
    end % end of if statement of the 5th month.
```

```
if currentmonth == 6
r6 = load(['zfilter' strftime(st, '%Y') '0601_' ...
          strftime(st, '%Y') '0701.gn']);
end % end of if statement of the 6th month.

if currentmonth == 7
r7 = load(['zfilter' strftime(st, '%Y') '0701_' ...
          strftime(st, '%Y') '0801.gn']);
end % end of if statement of the 7th month.

if currentmonth == 8
r8 = load(['zfilter' strftime(st, '%Y') '0801_' ...
          strftime(st, '%Y') '0901.gn']);
end % end of if statement of the 8th month.

if currentmonth == 9
r9 = load(['zfilter' strftime(st, '%Y') '0901_' ...
          strftime(st, '%Y') '1001.gn']);
end % end of if statement of the 9th month.

if currentmonth == 10
r10 = load(['zfilter' strftime(st, '%Y') '1001_' ...
           strftime(st, '%Y') '1101.gn']);
end % end of if statement of the 10th month.

if currentmonth == 11
r11 = load(['zfilter' strftime(st, '%Y') '1101_' ...
           strftime(st, '%Y') '1201.gn']);
end % end of if statement of the 11th month.

if currentmonth == 12
r12 = load(['zfilter' strftime(st, '%Y') '1201_' ...
           strftime(ed, '%Y') '0101.gn']);
end % end of if statement of the 12th month.
```

```
end % end of for loop for the months.

% collect all the results for all the months.
results = [
    r1
    r2
    r3
    r4
    r5
    r6
    r7
    r8
    r9
    r10
    r11
    r12
];

% find the size of the total results of the whole year.
results_size = size(results);

% specify the name of the catalogue that will be created.
catalogue_name = char(['zcatalogue' strftime(st, '%Y')
    '.gn']);

% open a file to save the total results.
% the results are saved in ASCII format.
% if the function is called for the same year,
% the contents of the results file will be overwritten.
fid_results = fopen(catalogue_name, 'w');

% initialise a counter that will keep the
% position of the data in the total results matrix.
results_counter = 1;

% start from the begin of the total results matrix and
% stop at the end of the total results matrix.
```



```
while results_counter <= length(results)

    fprintf(fid_results, ...
        '%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %.2f
        %.2f\n', ...
        results(results_counter, 1), ...
        results(results_counter, 2), ...
        results(results_counter, 3), ...
        results(results_counter, 4), ...
        results(results_counter, 5), ...
        results(results_counter, 6), ...
        results(results_counter, 7), ...
        results(results_counter, 8), ...
        results(results_counter, 9), ...
        results(results_counter, 10), ...
        results(results_counter, 11), ...
        results(results_counter, 12), ...
        results(results_counter, 13), ...
        results(results_counter, 14), ...
        results(results_counter, 15), ...
        results(results_counter, 16), ...
        results(results_counter, 17), ...
        results(results_counter, 18), ...
        results(results_counter, 19), ...
        results(results_counter, 20));

    % go to the next sample of the data_values_counter.
    results_counter = results_counter + 1;
end % end of while loop for the total results.

% close the file for the total results.
status = fclose(fid_results);
```

dbaseiris.m

```
function dbaseiris(year, loadmonths, statanalysis, filename)

%DBASEIRIS
%
%Date Started: 09/07/2000 2:42pm
%Date Completed: 17/07/2000 8:10pm
%Creator: G.Nikitas
%DCS, Lancaster University
%
% the program loads the contents of the results file depending
on
% the user's inputs.
% then it searches the results file (data base search) and
% returns to the user the requested output.
%
% the user should specify the following inputs:
% year - for the year he wants to apply the analysis. e.g.
1995
% loadmonths - for the months he is interested. e.g. [4 7
10:12]
% statanalysis - for the type of analysis he wants to apply.
% e.g. 'all', 'spike', 'aa', 'pca', 'noise' or 'datagap'
% filename - where the dbase search results will be saved.
% e.g. 'myfile' the default filename is 'tmpdbase.gn'

% set the default name for the dbase results file.
if nargin < 4
    filename = 'tmpdbase.gn';
end

% set the default event for the statistical analysis.
if nargin < 3
    statanalysis = 'all';
end
```

```
% set the default value for the months.
if nargin < 2
    loadmonths = [1:12];
end

% open a file to save the results
% of the dbase analysis.
fid_dbase = fopen(filename, 'w');

% print a message into the file.
fprintf(fid_dbase, ...
    'Data Base Server v1.3 - Process Requested on %s\n',
    date); ...

% print a message on the Matlab Workspace.
fprintf('\nData Base Search Process. Please Wait!\n');

% six different cases for the type of the statistical
analysis.
stattype1 = 'all';
stattype2 = 'spike';
stattype3 = 'aa';
stattype4 = 'pca';
stattype5 = 'noise';
stattype6 = 'datagap';

% check if the correct input for the statistical analysis has
been entered.
if ~strcmp(statanalysis, stattype1) & ...
    ~strcmp(statanalysis, stattype2) & ...
    ~strcmp(statanalysis, stattype3) & ...
    ~strcmp(statanalysis, stattype4) & ...
    ~strcmp(statanalysis, stattype5) & ...
```

```
~strcmp(statanalysis, stattype6)

% print a message on the workspace.
fprintf('The input %s is not valid.\n', statanalysis);
fprintf('valid inputs: all, spike, aa, pca, noise,
datagap\n');

% exit the function and return to the matlab workspace.
return;

end % end of if statement for the input of the statistical
analysis.

% go through all the months, take one each time.
for currentmonth = loadmonths

% try to load and analyse the following results file, if
failed to
% to load any file go to catch (try to load the ext file).
try

% if the month you entered is between 1 and 11
% then execute this if statement.
if ((currentmonth == 1 | currentmonth == 2 | currentmonth == 3
| currentmonth == 4 | ...
    currentmonth == 5 | currentmonth == 6 | currentmonth ==
7 | currentmonth == 8 | ...
    currentmonth == 9 | currentmonth == 10 | currentmonth ==
11) ...
    & (year >= 1994))

% specify the general characteristics of the results files.
nextyear = year + 1;
nextmonth = currentmonth + 1;
results_starttime = time([year currentmonth 01 0 0 0]);
results_endtime = time([year nextmonth 01 0 0 0]);
```

```
% the name of the files for the results depends on the
% requested year and month, that have been given by the user.
resultsfile = ...
    char(['zfilter' ...
        strftime(results_starttime,'%Y') ...
        strftime(results_starttime,'%m') '01_' ...
        strftime(results_endtime,'%Y') ...
        strftime(results_endtime,'%m') ...
        '01.gn']);

% if the month you entered is between 12
% then execute this elseif statement.
elseif (currentmonth == 12 & year >= 1994)

% specify the general characteristics of the results files.
nextyear = year + 1;
nextmonth = currentmonth + 1;
results_starttime = time([year currentmonth 01 0 0 0]);
results_endtime = time([nextyear 01 01 0 0 0]);

% the name of the files for the results depends on the
% requested year and month, that have been given by the user.
resultsfile = ...
    char(['zfilter' ...
        strftime(results_starttime,'%Y') ...
        strftime(results_starttime,'%m') '01_' ...
        strftime(results_endtime,'%Y') ...
        strftime(results_endtime,'%m') ...
        '01.gn']);

end % end of if-else statement.

% load the all files contains the results of the data analysis
for
% the requested year.
results = load([resultsfile]);
```

```
% find the size of the results matrix.
results_size = size(results);

% initialise the counters that keep track of the number
% of different events that occur in one month.
counter_aa_spikes = 0;
counter_noise = 0;
counter_aa_all = 0;
counter_noise = 0;
counter_datagap = 0;
counter_pca = 0;
counter_noise = 0;
counter_datagap = 0;

% initialise the counters that keep track of the useful
% information for the different events.

% Spikes AAs.
aa_spikes_startyear = 0;
aa_spikes_startmonth = 0;
aa_spikes_startday = 0;
aa_spikes_starthour = 0;
aa_spikes_startmin = 0;
aa_spikes_startsec = 0;
aa_spikes_endyear = 0;
aa_spikes_endmonth = 0;
aa_spikes_endday = 0;
aa_spikes_endhour = 0;
aa_spikes_endmin = 0;
aa_spikes_endsec = 0;
aa_spikes_durday = 0;
aa_spikes_durhour = 0;
aa_spikes_durmin = 0;
aa_spikes_dursec = 0;
aa_spikes_max = 0;
```

```
aa_spikes_mean = 0;
```

```
% All AAs.
```

```
aa_all_startyear = 0;  
aa_all_startmonth = 0;  
aa_all_startday = 0;  
aa_all_starthour = 0;  
aa_all_startmin = 0;  
aa_all_startsec = 0;  
aa_all_endyear = 0;  
aa_all_endmonth = 0;  
aa_all_endday = 0;  
aa_all_endhour = 0;  
aa_all_endmin = 0;  
aa_all_endsec = 0;  
aa_all_durday = 0;  
aa_all_durhour = 0;  
aa_all_durmin = 0;  
aa_all_dursec = 0;  
aa_all_max = 0;  
aa_all_mean = 0;
```

```
% PCAs.
```

```
pca_startyear = 0;  
pca_startmonth = 0;  
pca_startday = 0;  
pca_starthour = 0;  
pca_startmin = 0;  
pca_startsec = 0;  
pca_endyear = 0;  
pca_endmonth = 0;  
pca_endday = 0;  
pca_endhour = 0;  
pca_endmin = 0;  
pca_endsec = 0;  
pca_durday = 0;  
pca_durhour = 0;
```

```
pca_durmin = 0;
pca_dursec = 0;
pca_max = 0;
pca_mean = 0;

% NOISE.
noise_startyear = 0;
noise_startmonth = 0;
noise_startday = 0;
noise_starthour = 0;
noise_startmin = 0;
noise_startsec = 0;
noise_endyear = 0;
noise_endmonth = 0;
noise_endday = 0;
noise_endhour = 0;
noise_endmin = 0;
noise_endsec = 0;
noise_durday = 0;
noise_durhour = 0;
noise_durmin = 0;
noise_dursec = 0;
noise_max = 0;
noise_mean = 0;

% DataGaps.
datagap_startyear = 0;
datagap_startmonth = 0;
datagap_startday = 0;
datagap_starthour = 0;
datagap_startmin = 0;
datagap_startsec = 0;
datagap_endyear = 0;
datagap_endmonth = 0;
datagap_endday = 0;
datagap_endhour = 0;
datagap_endmin = 0;
```



```
datagap_endsec = 0;
datagap_durday = 0;
datagap_durhour = 0;
datagap_durmin = 0;
datagap_dursec = 0;
datagap_max = 0;
datagap_mean = 0;

% initialise the variables that keep the total number
% of different events that occur every month for one year.
total_pca = zeros(1, 12);
total_noise = zeros(1, 12);
total_datagap = zeros(1, 12);
total_aa_spikes = zeros(1, 12);
total_aa_all = zeros(1, 12);
total_pca = zeros(1, 12);
total_noise = zeros(1, 12);
total_datagap = zeros(1, 12);

% initialise a counter for the while loop below.
results_counter = 1;

% start from the begin of the analysis results until
% the last result.
while results_counter <= (results_size(1))

% continue the search process if each results is from the
widebeam (50).
if (results(results_counter, 2) == 50) |
isnan(results(results_counter, 2))
```

```
% find the SPIKES - AURORAL ABSORPTIONS.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 16) == 0) & ...
    (results(results_counter, 17) <= 6) & ...
    (results(results_counter, 4) == currentmonth)

    % collect useful information.
    % start and end time, duration, max and mean values.
    aa_spikes_startyear = results(results_counter, 3);
    aa_spikes_startmonth = results(results_counter, 4);
    aa_spikes_startday = results(results_counter, 5);
    aa_spikes_starthour = results(results_counter, 6);
    aa_spikes_startmin = results(results_counter, 7);
    aa_spikes_startsec = results(results_counter, 8);
    aa_spikes_endyear = results(results_counter, 9);
    aa_spikes_endmonth = results(results_counter, 10);
    aa_spikes_endday = results(results_counter, 11);
    aa_spikes_endhour = results(results_counter, 12);
    aa_spikes_endmin = results(results_counter, 13);
    aa_spikes_endsec = results(results_counter, 14);
    aa_spikes_durday = results(results_counter, 15);
    aa_spikes_durhour = results(results_counter, 16);
    aa_spikes_durmin = results(results_counter, 17);
    aa_spikes_dursec = results(results_counter, 18);
    aa_spikes_max = results(results_counter, 19);
    aa_spikes_mean = results(results_counter, 20);

    % the if statement is true, increament the counter by one.
    counter_aa_spikes = counter_aa_spikes + 1;

end % end of if statement that looks for the SPIKES - AAs.

% find ALL the AURORAL ABSORPTIONS.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 4) == currentmonth)
```

```
% collect useful information.
% start and end time, duration, max and mean values.
aa_all_startyear = results(results_counter, 3);
aa_all_startmonth = results(results_counter, 4);
aa_all_startday = results(results_counter, 5);
aa_all_starthour = results(results_counter, 6);
aa_all_startmin = results(results_counter, 7);
aa_all_startsec = results(results_counter, 8);
aa_all_endyear = results(results_counter, 9);
aa_all_endmonth = results(results_counter, 10);
aa_all_endday = results(results_counter, 11);
aa_all_endhour = results(results_counter, 12);
aa_all_endmin = results(results_counter, 13);
aa_all_endsec = results(results_counter, 14);
aa_all_durday = results(results_counter, 15);
aa_all_durhour = results(results_counter, 16);
aa_all_durmin = results(results_counter, 17);
aa_all_dursec = results(results_counter, 18);
aa_all_max = results(results_counter, 19);
aa_all_mean = results(results_counter, 20);

% the if statement is true, increament the counter by one.
counter_aa_all = counter_aa_all + 1;

end % end of if statement that looks for the ALL - AAs.

% find the POLAR CAP ABSORPTIONS.
if (results(results_counter, 1) == 5) & ...
    (results(results_counter, 4) == currentmonth)

    % collect useful information.
    % start and end time, duration, max and mean values.
    pca_startyear = results(results_counter, 3);
    pca_startmonth = results(results_counter, 4);
    pca_startday = results(results_counter, 5);
    pca_starthour = results(results_counter, 6);
```

```
pca_startmin = results(results_counter, 7);
pca_startsec = results(results_counter, 8);
pca_endyear = results(results_counter, 9);
pca_endmonth = results(results_counter, 10);
pca_endday = results(results_counter, 11);
pca_endhour = results(results_counter, 12);
pca_endmin = results(results_counter, 13);
pca_endsec = results(results_counter, 14);
pca_durday = results(results_counter, 15);
pca_durhour = results(results_counter, 16);
pca_durmin = results(results_counter, 17);
pca_dursec = results(results_counter, 18);
pca_max = results(results_counter, 19);
pca_mean = results(results_counter, 20);

% the if statement is true, increament the counter by one.
counter_pca = counter_pca + 1;

end % end of if statement that looks for the PCAs.

% find the NOISE.
if (results(results_counter, 1) == 3) & ...
    (results(results_counter, 4) == currentmonth)

    % collect useful information.
    % start and end time, duration, max and mean values.
    noise_startyear = results(results_counter, 3);
    noise_startmonth = results(results_counter, 4);
    noise_startday = results(results_counter, 5);
    noise_starthour = results(results_counter, 6);
    noise_startmin = results(results_counter, 7);
    noise_startsec = results(results_counter, 8);
    noise_endyear = results(results_counter, 9);
    noise_endmonth = results(results_counter, 10);
    noise_endday = results(results_counter, 11);
    noise_endhour = results(results_counter, 12);
```

```
noise_endmin = results(results_counter, 13);
noise_endsec = results(results_counter, 14);
noise_durday = results(results_counter, 15);
noise_durhour = results(results_counter, 16);
noise_durmin = results(results_counter, 17);
noise_dursec = results(results_counter, 18);
noise_max = results(results_counter, 19);
noise_mean = results(results_counter, 20);

% the if statement is true, increament the counter by one.
counter_noise = counter_noise + 1;

end % end of if statement that looks for the NOISE.

% find the DATAGAPS.
if (results(results_counter, 1) == 2) & ...
    (results(results_counter, 4) == currentmonth)

    % collect useful information.
    % start and end time, duration, max and mean values.
    datagap_startyear = results(results_counter, 3);
    datagap_startmonth = results(results_counter, 4);
    datagap_startday = results(results_counter, 5);
    datagap_starthour = results(results_counter, 6);
    datagap_startmin = results(results_counter, 7);
    datagap_startsec = results(results_counter, 8);
    datagap_endyear = results(results_counter, 9);
    datagap_endmonth = results(results_counter, 10);
    datagap_endday = results(results_counter, 11);
    datagap_endhour = results(results_counter, 12);
    datagap_endmin = results(results_counter, 13);
    datagap_endsec = results(results_counter, 14);
    datagap_durday = results(results_counter, 15);
    datagap_durhour = results(results_counter, 16);
    datagap_durmin = results(results_counter, 17);
    datagap_dursec = results(results_counter, 18);
```

```
datagap_max = results(results_counter, 19);
datagap_mean = results(results_counter, 20);

% the if statement is true, increament the counter by one.
counter_datagap = counter_datagap + 1;

end % end of if statement that looks for the DATAGAPs.

% statistical analysis no:1.
if strcmp(statanalysis, stattype1)

    % collect all (the total) the useful information.

    total_aa_spikes_startyear(counter_aa_spikes+1) =
aa_spikes_startyear;
    total_aa_spikes_startmonth(counter_aa_spikes+1) =
aa_spikes_startmonth;
    total_aa_spikes_startday(counter_aa_spikes+1) =
aa_spikes_startday;
    total_aa_spikes_starthour(counter_aa_spikes+1) =
aa_spikes_starthour;
    total_aa_spikes_startmin(counter_aa_spikes+1) =
aa_spikes_startmin;
    total_aa_spikes_startsec(counter_aa_spikes+1) =
aa_spikes_startsec;
    total_aa_spikes_endyear(counter_aa_spikes+1) =
aa_spikes_endyear;
    total_aa_spikes_endmonth(counter_aa_spikes+1) =
aa_spikes_endmonth;
    total_aa_spikes_endday(counter_aa_spikes+1) =
aa_spikes_endday;
    total_aa_spikes_endhour(counter_aa_spikes+1) =
aa_spikes_endhour;
    total_aa_spikes_endmin(counter_aa_spikes+1) =
aa_spikes_endmin;
```

```
total_aa_spikes_endsec(counter_aa_spikes+1) =
aa_spikes_endsec;
total_aa_spikes_durday(counter_aa_spikes+1) =
aa_spikes_durday;
total_aa_spikes_durhour(counter_aa_spikes+1) =
aa_spikes_durhour;
total_aa_spikes_durmin(counter_aa_spikes+1) =
aa_spikes_durmin;
total_aa_spikes_dursec(counter_aa_spikes+1) =
aa_spikes_dursec;
total_aa_spikes_max(counter_aa_spikes+1) = aa_spikes_max;
total_aa_spikes_mean(counter_aa_spikes+1) =
aa_spikes_mean;

total_aa_all_startyear(counter_aa_all+1) =
aa_all_startyear;
total_aa_all_startmonth(counter_aa_all+1) =
aa_all_startmonth;
total_aa_all_startday(counter_aa_all+1) =
aa_all_startday;
total_aa_all_starthour(counter_aa_all+1) =
aa_all_starthour;
total_aa_all_startmin(counter_aa_all+1) = aa_all_startmin;
total_aa_all_startsec(counter_aa_all+1) = aa_all_startsec;
total_aa_all_endyear(counter_aa_all+1) = aa_all_endyear;
total_aa_all_endmonth(counter_aa_all+1) = aa_all_endmonth;
total_aa_all_endday(counter_aa_all+1) = aa_all_endday;
total_aa_all_endhour(counter_aa_all+1) = aa_all_endhour;
total_aa_all_endmin(counter_aa_all+1) = aa_all_endmin;
total_aa_all_endsec(counter_aa_all+1) = aa_all_endsec;
total_aa_all_durday(counter_aa_all+1) = aa_all_durday;
total_aa_all_durhour(counter_aa_all+1) = aa_all_durhour;
total_aa_all_durmin(counter_aa_all+1) = aa_all_durmin;
total_aa_all_dursec(counter_aa_all+1) = aa_all_dursec;
total_aa_all_max(counter_aa_all+1) = aa_all_max;
total_aa_all_mean(counter_aa_all+1) = aa_all_mean;
```

```
total_pca_startyear(counter_pca+1) = pca_startyear;
total_pca_startmonth(counter_pca+1) = pca_startmonth;
total_pca_startday(counter_pca+1) = pca_startday;
total_pca_starthour(counter_pca+1) = pca_starthour;
total_pca_startmin(counter_pca+1) = pca_startmin;
total_pca_startsec(counter_pca+1) = pca_startsec;
total_pca_endyear(counter_pca+1) = pca_endyear;
total_pca_endmonth(counter_pca+1) = pca_endmonth;
total_pca_endday(counter_pca+1) = pca_endday;
total_pca_endhour(counter_pca+1) = pca_endhour;
total_pca_endmin(counter_pca+1) = pca_endmin;
total_pca_endsec(counter_pca+1) = pca_endsec;
total_pca_durday(counter_pca+1) = pca_durday;
total_pca_durhour(counter_pca+1) = pca_durhour;
total_pca_durmin(counter_pca+1) = pca_durmin;
total_pca_dursec(counter_pca+1) = pca_dursec;
total_pca_max(counter_pca+1) = pca_max;
total_pca_mean(counter_pca+1) = pca_mean;

total_noise_startyear(counter_noise+1) = noise_startyear;
total_noise_startmonth(counter_noise+1) =
noise_startmonth;
total_noise_startday(counter_noise+1) = noise_startday;
total_noise_starthour(counter_noise+1) = noise_starthour;
total_noise_startmin(counter_noise+1) = noise_startmin;
total_noise_startsec(counter_noise+1) = noise_startsec;
total_noise_endyear(counter_noise+1) = noise_endyear;
total_noise_endmonth(counter_noise+1) = noise_endmonth;
total_noise_endday(counter_noise+1) = noise_endday;
total_noise_endhour(counter_noise+1) = noise_endhour;
total_noise_endmin(counter_noise+1) = noise_endmin;
total_noise_endsec(counter_noise+1) = noise_endsec;
total_noise_durday(counter_noise+1) = noise_durday;
total_noise_durhour(counter_noise+1) = noise_durhour;
total_noise_durmin(counter_noise+1) = noise_durmin;
total_noise_dursec(counter_noise+1) = noise_dursec;
total_noise_max(counter_noise+1) = noise_max;
```



```
total_noise_mean(counter_noise+1) = noise_mean;

total_datagap_startyear(counter_datagap+1) =
datagap_startyear;
total_datagap_startmonth(counter_datagap+1) =
datagap_startmonth;
total_datagap_startday(counter_datagap+1) =
datagap_startday;
total_datagap_starthour(counter_datagap+1) =
datagap_starthour;
total_datagap_startmin(counter_datagap+1) =
datagap_startmin;
total_datagap_startsec(counter_datagap+1) =
datagap_startsec;
total_datagap_endyear(counter_datagap+1) =
datagap_endyear;
total_datagap_endmonth(counter_datagap+1) =
datagap_endmonth;
total_datagap_endday(counter_datagap+1) = datagap_endday;
total_datagap_endhour(counter_datagap+1) =
datagap_endhour;
total_datagap_endmin(counter_datagap+1) = datagap_endmin;
total_datagap_endsec(counter_datagap+1) = datagap_endsec;
total_datagap_durday(counter_datagap+1) = datagap_durday;
total_datagap_durhour(counter_datagap+1) =
datagap_durhour;
total_datagap_durmin(counter_datagap+1) = datagap_durmin;
total_datagap_dursec(counter_datagap+1) = datagap_dursec;
total_datagap_max(counter_datagap+1) = datagap_max;
total_datagap_mean(counter_datagap+1) = datagap_mean;

% collect the total number for each event.
total_aa_spikes(currentmonth) = counter_aa_spikes;
total_aa_all(currentmonth) = counter_aa_all;
total_pca(currentmonth) = counter_pca;
total_noise(currentmonth) = counter_noise;
```

```
total_datagap(currentmonth) = counter_datagap;

end % end of if statement for the statistical analysis no:1.

% statistical analysis no:2.
if strcmp(statanalysis, stattype2)

    % collect all (the total) the useful information.
    total_aa_spikes_startyear(counter_aa_spikes+1) =
aa_spikes_startyear;
    total_aa_spikes_startmonth(counter_aa_spikes+1) =
aa_spikes_startmonth;
    total_aa_spikes_startday(counter_aa_spikes+1) =
aa_spikes_startday;
    total_aa_spikes_starthour(counter_aa_spikes+1) =
aa_spikes_starthour;
    total_aa_spikes_startmin(counter_aa_spikes+1) =
aa_spikes_startmin;
    total_aa_spikes_startsec(counter_aa_spikes+1) =
aa_spikes_startsec;
    total_aa_spikes_endyear(counter_aa_spikes+1) =
aa_spikes_endyear;
    total_aa_spikes_endmonth(counter_aa_spikes+1) =
aa_spikes_endmonth;
    total_aa_spikes_endday(counter_aa_spikes+1) =
aa_spikes_endday;
    total_aa_spikes_endhour(counter_aa_spikes+1) =
aa_spikes_endhour;
    total_aa_spikes_endmin(counter_aa_spikes+1) =
aa_spikes_endmin;
    total_aa_spikes_endsec(counter_aa_spikes+1) =
aa_spikes_endsec;
    total_aa_spikes_durday(counter_aa_spikes+1) =
aa_spikes_durday;
    total_aa_spikes_durhour(counter_aa_spikes+1) =
aa_spikes_durhour;
```

```
        total_aa_spikes_durmin(counter_aa_spikes+1) =  
aa_spikes_durmin;  
        total_aa_spikes_dursecc(counter_aa_spikes+1) =  
aa_spikes_dursecc;  
        total_aa_spikes_max(counter_aa_spikes+1) = aa_spikes_max;  
        total_aa_spikes_mean(counter_aa_spikes+1) =  
aa_spikes_mean;  
  
        % collect the total number of SPIKES - AAs.  
        total_aa_spikes(currentmonth) = counter_aa_spikes;  
  
end % end of if statement for the statistical analysis no:2.  
  
% statistical analysis no:3.  
if strcmp(statanalysis, stattype3)  
  
        % collect all (the total) the useful information.  
        total_aa_all_startyear(counter_aa_all+1) =  
aa_all_startyear;  
        total_aa_all_startmonth(counter_aa_all+1) =  
aa_all_startmonth;  
        total_aa_all_startday(counter_aa_all+1) =  
aa_all_startday;  
        total_aa_all_starthour(counter_aa_all+1) =  
aa_all_starthour;  
        total_aa_all_startmin(counter_aa_all+1) = aa_all_startmin;  
        total_aa_all_startsec(counter_aa_all+1) = aa_all_startsec;  
        total_aa_all_endyear(counter_aa_all+1) = aa_all_endyear;  
        total_aa_all_endmonth(counter_aa_all+1) = aa_all_endmonth;  
        total_aa_all_endday(counter_aa_all+1) = aa_all_endday;  
        total_aa_all_endhour(counter_aa_all+1) = aa_all_endhour;  
        total_aa_all_endmin(counter_aa_all+1) = aa_all_endmin;  
        total_aa_all_endsec(counter_aa_all+1) = aa_all_endsec;  
        total_aa_all_durday(counter_aa_all+1) = aa_all_durday;  
        total_aa_all_durhour(counter_aa_all+1) = aa_all_durhour;  
        total_aa_all_durmin(counter_aa_all+1) = aa_all_durmin;
```

```
total_aa_all_dursec(counter_aa_all+1) = aa_all_dursec;
total_aa_all_max(counter_aa_all+1) = aa_all_max;
total_aa_all_mean(counter_aa_all+1) = aa_all_mean;

% collect the total number of ALL AAs.
total_aa_all(currentmonth) = counter_aa_all;

end % end of if statement for the statistical analysis no:3.

% statistical analysis no:4.
if strcmp(statanalysis, stattype4)

    % collect all (the total) the useful information.
    total_pca_startyear(counter_pca+1) = pca_startyear;
    total_pca_startmonth(counter_pca+1) = pca_startmonth;
    total_pca_startday(counter_pca+1) = pca_startday;
    total_pca_starthour(counter_pca+1) = pca_starthour;
    total_pca_startmin(counter_pca+1) = pca_startmin;
    total_pca_startsec(counter_pca+1) = pca_startsec;
    total_pca_endyear(counter_pca+1) = pca_endyear;
    total_pca_endmonth(counter_pca+1) = pca_endmonth;
    total_pca_endday(counter_pca+1) = pca_endday;
    total_pca_endhour(counter_pca+1) = pca_endhour;
    total_pca_endmin(counter_pca+1) = pca_endmin;
    total_pca_endsec(counter_pca+1) = pca_endsec;
    total_pca_durday(counter_pca+1) = pca_durday;
    total_pca_durhour(counter_pca+1) = pca_durhour;
    total_pca_durmin(counter_pca+1) = pca_durmin;
    total_pca_dursec(counter_pca+1) = pca_dursec;
    total_pca_max(counter_pca+1) = pca_max;
    total_pca_mean(counter_pca+1) = pca_mean;

    % collect the total number of PCAs.
    total_pca(currentmonth) = counter_pca;

end % end of if statement for the statistical analysis no:4.
```

```
% statistical analysis no:5.
if strcmp(statanalysis, stattype5)

    % collect all (the total) the useful information.
    total_noise_startyear(counter_noise+1) = noise_startyear;
    total_noise_startmonth(counter_noise+1) =
noise_startmonth;
    total_noise_startday(counter_noise+1) = noise_startday;
    total_noise_starthour(counter_noise+1) = noise_starthour;
    total_noise_startmin(counter_noise+1) = noise_startmin;
    total_noise_startsec(counter_noise+1) = noise_startsec;
    total_noise_endyear(counter_noise+1) = noise_endyear;
    total_noise_endmonth(counter_noise+1) = noise_endmonth;
    total_noise_endday(counter_noise+1) = noise_endday;
    total_noise_endhour(counter_noise+1) = noise_endhour;
    total_noise_endmin(counter_noise+1) = noise_endmin;
    total_noise_endsec(counter_noise+1) = noise_endsec;
    total_noise_durday(counter_noise+1) = noise_durday;
    total_noise_durhour(counter_noise+1) = noise_durhour;
    total_noise_durmin(counter_noise+1) = noise_durmin;
    total_noise_dursec(counter_noise+1) = noise_dursec;
    total_noise_max(counter_noise+1) = noise_max;
    total_noise_mean(counter_noise+1) = noise_mean;

    % collect the total number of NOISE.
    total_noise(currentmonth) = counter_noise;

end % end of if statement for the statistical analysis no:5.

% statistical analysis no:6.
if strcmp(statanalysis, stattype6)

    % collect all (the total) the useful information.
    total_datagap_startyear(counter_datagap+1) =
```

```
datagap_startyear;
    total_datagap_startmonth(counter_datagap+1) =
datagap_startmonth;
    total_datagap_startday(counter_datagap+1) =
datagap_startday;
    total_datagap_starthour(counter_datagap+1) =
datagap_starthour;
    total_datagap_startmin(counter_datagap+1) =
datagap_startmin;
    total_datagap_startsec(counter_datagap+1) =
datagap_startsec;
    total_datagap_endyear(counter_datagap+1) =
datagap_endyear;
    total_datagap_endmonth(counter_datagap+1) =
datagap_endmonth;
    total_datagap_endday(counter_datagap+1) = datagap_endday;
    total_datagap_endhour(counter_datagap+1) =
datagap_endhour;
    total_datagap_endmin(counter_datagap+1) = datagap_endmin;
    total_datagap_endsec(counter_datagap+1) = datagap_endsec;
    total_datagap_durday(counter_datagap+1) = datagap_durday;
    total_datagap_durhour(counter_datagap+1) =
datagap_durhour;
    total_datagap_durmin(counter_datagap+1) = datagap_durmin;
    total_datagap_dursec(counter_datagap+1) = datagap_dursec;
    total_datagap_max(counter_datagap+1) = datagap_max;
    total_datagap_mean(counter_datagap+1) = datagap_mean;

    % collect the total number of DATAGAPs.
    total_datagap(currentmonth) = counter_datagap;

end % end of if statement for the statistical analysis no:6.

else
```

```
% print a message on the workspace.
fprintf('\nWarning!\n');
fprintf('Only the widebeam is processed by the Data Base
Search.\n');
fprintf('The file %s contains results from the beam %d as
well.\n', ...
        resultsfile, results(results_counter, 2));
fprintf('\nProcess Continues. Please Wait!\n');

end % end of if-else statement that checks for the widebeam.


% increament the counter by one.
results_counter = results_counter + 1;

end % end of while loop for the results.


% reinitialise the counters that keep track of the number
% of different events that occur in one month.
counter_aa_spikes = 0;
counter_aa_all = 0;
counter_pca = 0;
counter_noise = 0;
counter_datagap = 0;


% each variable is an array 1x12 and in each column keeps the
% number of occrrences of a single event.
total_aa_spikes;
total_aa_all;
total_pca;
total_noise;
total_datagap;


% these information will be needed later for the histograms.
```

```

totalmonths = [1 2 3 4 5 6 7 8 9 10 11 12];

% all the events.
if strcmp(statanalysis, stattype1)

% put the total number of Spiky AAs, the total number of All
AAs,
% the total number of PCAs, the total number of Noise and the
% total number of Datagaps into a single array.
total_all = ...
    [total_aa_spikes; total_aa_all; ...
    total_pca; total_noise; total_datagap];

% print useful information into the file.
fprintf(fid_dbase, '\n\nData Base Search Results for ALL
Events\n');
fprintf(fid_dbase, 'Year : %d\n', year);
fprintf(fid_dbase, 'Month: %d\n', currentmonth);
fprintf(fid_dbase, 'Beam : Widebeam\n');
fprintf(fid_dbase, ...
    '\nTotal Number of Spikes                :
%d', sum(total_aa_spikes));
fprintf(fid_dbase, ...
    '\nTotal Number of All the Auroral Absorptions :
%d', sum(total_aa_all));
fprintf(fid_dbase, ...
    '\nTotal Number of Polar Cap Absorptions      :
%d', sum(total_pca));
fprintf(fid_dbase, ...
    '\nTotal Number of Noise                      :
%d', sum(total_noise));
fprintf(fid_dbase, ...
    '\nTotal Number of Data Gaps                  :
%d\n', sum(total_datagap));

```



```

% start the search for the Spikes.
fprintf(fid_dbase, '\nSearch Results for Spikes\n');

% go through all the requested event.
for y = 2:total_aa_spikes(:, currentmonth)+1

    fprintf(fid_dbase, '\nSpike Event no.%d', y-1);
    fprintf(fid_dbase, '\nStart Time Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
        total_aa_spikes_startday(y), ...
        total_aa_spikes_startmonth(y), ...
        total_aa_spikes_startyear(y), ...
        total_aa_spikes_starthour(y), ...
        total_aa_spikes_startmin(y), ...
        total_aa_spikes_startsec(y));
    fprintf(fid_dbase, '\nEnd Time   Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
        total_aa_spikes_endday(y), ...
        total_aa_spikes_endmonth(y), ...
        total_aa_spikes_endyear(y), ...
        total_aa_spikes_endhour(y), ...
        total_aa_spikes_endmin(y), ...
        total_aa_spikes_endsec(y));
    fprintf(fid_dbase, '\nDuration   %ddays and %.2dh %.2dm %.2ds',
...
        total_aa_spikes_durday(y), ...
        total_aa_spikes_durhour(y), ...
        total_aa_spikes_durmin(y), ...
        total_aa_spikes_dursec(y));
    fprintf(fid_dbase, '\nMax           %.2fdB', ...
        total_aa_spikes_max(y));
    fprintf(fid_dbase, '\nMean           %.2fdB\n', ...
        total_aa_spikes_mean(y));

end % end of for loop for the information.

```

```

% start the search for All the AAs.
fprintf(fid_dbase, '\nSearch Results for All the Auroral
Absorptions\n');

% go through all the requested event.
for y = 2:total_aa_all(:, currentmonth)+1

fprintf(fid_dbase, '\nAuroral Event no.%d', y-1);
fprintf(fid_dbase, '\nStart Time Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
    total_aa_all_startday(y), ...
    total_aa_all_startmonth(y), ...
    total_aa_all_startyear(y), ...
    total_aa_all_starthour(y), ...
    total_aa_all_startmin(y), ...
    total_aa_all_startsec(y));
fprintf(fid_dbase, '\nEnd Time    Date %d/%d/%d Time
%.2d:%.2d:%.2d', ...
    total_aa_all_endday(y), ...
    total_aa_all_endmonth(y), ...
    total_aa_all_endyear(y), ...
    total_aa_all_endhour(y), ...
    total_aa_all_endmin(y), ...
    total_aa_all_endsec(y));
fprintf(fid_dbase, '\nDuration    %ddays and %.2dh %.2dm %.2ds',
...
    total_aa_all_durday(y), ...
    total_aa_all_durhour(y), ...
    total_aa_all_durmin(y), ...
    total_aa_all_dursec(y));
fprintf(fid_dbase, '\nMax          %.2fdB', ...
    total_aa_all_max(y));
fprintf(fid_dbase, '\nMean          %.2fdB\n', ...
    total_aa_all_mean(y));

end % end of for loop for the information.

```

```

% start the search for the PCAs.
fprintf(fid_dbase, '\nSearch Results for the Polar Cap
Absorptions\n');

% go through all the requested event.
for y = 2:total_pca(:, currentmonth)+1

fprintf(fid_dbase, '\nPCA Event no.%d', y-1);
fprintf(fid_dbase, '\nStart Time Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
    total_pca_startday(y), ...
    total_pca_startmonth(y), ...
    total_pca_startyear(y), ...
    total_pca_starthour(y), ...
    total_pca_startmin(y), ...
    total_pca_startsec(y));
fprintf(fid_dbase, '\nEnd Time    Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
    total_pca_endday(y), ...
    total_pca_endmonth(y), ...
    total_pca_endyear(y), ...
    total_pca_endhour(y), ...
    total_pca_endmin(y), ...
    total_pca_endsec(y));
fprintf(fid_dbase, '\nDuration    %ddays and %.2dh %.2dm %.2ds',
...
    total_pca_durday(y), ...
    total_pca_durhour(y), ...
    total_pca_durmin(y), ...
    total_pca_dursec(y));
fprintf(fid_dbase, '\nMax          %.2fdB', ...
    total_pca_max(y));
fprintf(fid_dbase, '\nMean          %.2fdB\n', ...
    total_pca_mean(y));

end % end of for loop for the information.

```

```

% start the search for the Noise.
fprintf(fid_dbase, '\nSearch Results for the Noise\n');

% go through all the requested event.
for y = 2:total_noise(:, currentmonth)+1

fprintf(fid_dbase, '\nNoise Event no.%d', y-1);
fprintf(fid_dbase, '\nStart Time Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
    total_noise_startday(y), ...
    total_noise_startmonth(y), ...
    total_noise_startyear(y), ...
    total_noise_starthour(y), ...
    total_noise_startmin(y), ...
    total_noise_startsec(y));
fprintf(fid_dbase, '\nEnd Time   Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
    total_noise_endday(y), ...
    total_noise_endmonth(y), ...
    total_noise_endyear(y), ...
    total_noise_endhour(y), ...
    total_noise_endmin(y), ...
    total_noise_endsec(y));
fprintf(fid_dbase, '\nDuration   %ddays and %.2dh %.2dm %.2ds',
...
    total_noise_durday(y), ...
    total_noise_durhour(y), ...
    total_noise_durmin(y), ...
    total_noise_dursec(y));
fprintf(fid_dbase, '\nMax           %.2fdB', ...
    total_noise_max(y));
fprintf(fid_dbase, '\nMean           %.2fdB\n', ...
    total_noise_mean(y));

end % end of for loop for the information.

```

```

% start the search for the DataGaps.
fprintf(fid_dbase, '\nSearch Results for the Data Gaps\n');

% go through all the requested event.
for y = 2:total_datagap(:, currentmonth)+1

fprintf(fid_dbase, '\nData Gap Event no.%d', y-1);
fprintf(fid_dbase, '\nStart Time Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
    total_datagap_startday(y), ...
    total_datagap_startmonth(y), ...
    total_datagap_startyear(y), ...
    total_datagap_starthour(y), ...
    total_datagap_startmin(y), ...
    total_datagap_startsec(y));
fprintf(fid_dbase, '\nEnd Time   Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
    total_datagap_endday(y), ...
    total_datagap_endmonth(y), ...
    total_datagap_endyear(y), ...
    total_datagap_endhour(y), ...
    total_datagap_endmin(y), ...
    total_datagap_endsec(y));
fprintf(fid_dbase, '\nDuration   %ddays and %.2dh %.2dm %.2ds',
...
    total_datagap_durday(y), ...
    total_datagap_durhour(y), ...
    total_datagap_durmin(y), ...
    total_datagap_dursec(y));
fprintf(fid_dbase, '\nMax           %.2fdB', ...
    total_datagap_max(y));
fprintf(fid_dbase, '\nMean           %.2fdB\n', ...
    total_datagap_mean(y));

end % end of for loop for the information.

```

```
end % end of the all the events.

% Spikes.
if strcmp(statanalysis, stattype2)

% put the total number of the Spiky AAs into a single array.
total_all = [total_aa_spikes];

% print useful information into the file.
fprintf(fid_dbase, '\n\nData Base Search Results for
Spikes\n');
fprintf(fid_dbase, 'Year : %d\n', year);
fprintf(fid_dbase, 'Month: %d\n', currentmonth);
fprintf(fid_dbase, 'Beam : Widebeam\n');
fprintf(fid_dbase, ...
        '\nTotal Number of Spikes : %d\n', sum(total_all));

% go through all the requested event.
for y = 2:total_all(:, currentmonth)+1

fprintf(fid_dbase, '\nSpike Event no.%d', y-1);
fprintf(fid_dbase, '\nStart Time Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
        total_aa_spikes_startday(y), ...
        total_aa_spikes_startmonth(y), ...
        total_aa_spikes_startyear(y), ...
        total_aa_spikes_starthour(y), ...
        total_aa_spikes_startmin(y), ...
        total_aa_spikes_startsec(y));
fprintf(fid_dbase, '\nEnd Time Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
        total_aa_spikes_endday(y), ...
        total_aa_spikes_endmonth(y), ...
        total_aa_spikes_endyear(y), ...
```

```

        total_aa_spikes_endhour(y), ...
        total_aa_spikes_endmin(y), ...
        total_aa_spikes_endsec(y));
fprintf(fid_dbase, '\nDuration    %ddays and %.2dh %.2dm %.2ds',
...
        total_aa_spikes_durday(y), ...
        total_aa_spikes_durhour(y), ...
        total_aa_spikes_durmin(y), ...
        total_aa_spikes_dursec(y));
fprintf(fid_dbase, '\nMax          %.2fdB', ...
        total_aa_spikes_max(y));
fprintf(fid_dbase, '\nMean          %.2fdB\n', ...
        total_aa_spikes_mean(y));

end % end of for loop for the information.

end % end of the Spikes.

% All the AAs.
if strcmp(statanalysis, stattype3)

% put the total number of All the AAs into a single array.
total_all = [total_aa_all];

% print useful information into the file.
fprintf(fid_dbase, '\n\n\nData Base Search Results for All the
Auroral Absorptions\n');
fprintf(fid_dbase, 'Year : %d\n', year);
fprintf(fid_dbase, 'Month: %d\n', currentmonth);
fprintf(fid_dbase, 'Beam : Widebeam\n');
fprintf(fid_dbase, ...
        '\nTotal Number of All the Auroral Absorptions :
%d\n', sum(total_all));

% go through all the requested event.
for y = 2:total_all(:, currentmonth)+1

```

```

fprintf(fid_dbase, '\nAuroral Event no.%d', y-1);
fprintf(fid_dbase, '\nStart Time Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
    total_aa_all_startday(y), ...
    total_aa_all_startmonth(y), ...
    total_aa_all_startyear(y), ...
    total_aa_all_starthour(y), ...
    total_aa_all_startmin(y), ...
    total_aa_all_startsec(y));
fprintf(fid_dbase, '\nEnd Time   Date %d/%d/%d Time
%.2d:%.2d:%.2d', ...
    total_aa_all_endday(y), ...
    total_aa_all_endmonth(y), ...
    total_aa_all_endyear(y), ...
    total_aa_all_endhour(y), ...
    total_aa_all_endmin(y), ...
    total_aa_all_endsec(y));
fprintf(fid_dbase, '\nDuration   %ddays and %.2dh %.2dm %.2ds',
...
    total_aa_all_durday(y), ...
    total_aa_all_durhour(y), ...
    total_aa_all_durmin(y), ...
    total_aa_all_dursec(y));
fprintf(fid_dbase, '\nMax           %.2fdB', ...
    total_aa_all_max(y));
fprintf(fid_dbase, '\nMean           %.2fdB\n', ...
    total_aa_all_mean(y));

end % end of for loop for the information.

end % end of all the AAs.

% PCAs.
if strcmp(statanalysis, stattype4)

```



```

% put the total number of the PCAs into a single array.
total_all = [total_pca];

% print useful information into the file.
fprintf(fid_dbase, '\n\nData Base Search Results for Polar
Cap Absorptions\n');
fprintf(fid_dbase, 'Year : %d\n', year);
fprintf(fid_dbase, 'Month: %d\n', currentmonth);
fprintf(fid_dbase, 'Beam : Widebeam\n');
fprintf(fid_dbase, ...
        '\nTotal Number of Polar Cap Absorptions :
%d\n', sum(total_all));

% go through all the requested event.
for y = 2:total_all(:, currentmonth)+1

fprintf(fid_dbase, '\nPCA Event no.%d', y-1);
fprintf(fid_dbase, '\nStart Time Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
        total_pca_startday(y), ...
        total_pca_startmonth(y), ...
        total_pca_startyear(y), ...
        total_pca_starthour(y), ...
        total_pca_startmin(y), ...
        total_pca_startsec(y));
fprintf(fid_dbase, '\nEnd Time    Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
        total_pca_endday(y), ...
        total_pca_endmonth(y), ...
        total_pca_endyear(y), ...
        total_pca_endhour(y), ...
        total_pca_endmin(y), ...
        total_pca_endsec(y));
fprintf(fid_dbase, '\nDuration    %ddays and %.2dh %.2dm %.2ds',
...
        total_pca_durday(y), ...
        total_pca_durhour(y), ...

```

```

        total_pca_durmin(y), ...
        total_pca_dursec(y));
fprintf(fid_dbase, '\nMax          %.2fdB', ...
        total_pca_max(y));
fprintf(fid_dbase, '\nMean          %.2fdB\n', ...
        total_pca_mean(y));

end % end of for loop for the information.

end % end of the PCAs.

% Noise.
if strcmp(statanalysis, stattype5)

% put the total number of the Noise into a single array.
total_all = [total_noise];

% print useful information into the file.
fprintf(fid_dbase, '\n\n\nData Base Search Results for
Noise\n');
fprintf(fid_dbase, 'Year : %d\n', year);
fprintf(fid_dbase, 'Month: %d\n', currentmonth);
fprintf(fid_dbase, 'Beam : Widebeam\n');
fprintf(fid_dbase, ...
        '\nTotal Number of Noise : %d\n', sum(total_all));

% go through all the requested event.
for y = 2:total_all(:, currentmonth)+1

fprintf(fid_dbase, '\nNoise Event no.%d', y-1);
fprintf(fid_dbase, '\nStart Time Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
        total_noise_startday(y), ...
        total_noise_startmonth(y), ...
        total_noise_startyear(y), ...
        total_noise_starthour(y), ...

```

```

        total_noise_startmin(y), ...
        total_noise_startsec(y));
fprintf(fid_dbase, '\nEnd Time    Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
        total_noise_endday(y), ...
        total_noise_endmonth(y), ...
        total_noise_endyear(y), ...
        total_noise_endhour(y), ...
        total_noise_endmin(y), ...
        total_noise_endsec(y));
fprintf(fid_dbase, '\nDuration    %ddays and %.2dh %.2dm %.2ds',
...
        total_noise_durday(y), ...
        total_noise_durhour(y), ...
        total_noise_durmin(y), ...
        total_noise_dursec(y));
fprintf(fid_dbase, '\nMax          %.2fdB', ...
        total_noise_max(y));
fprintf(fid_dbase, '\nMean          %.2fdB\n', ...
        total_noise_mean(y));

end % end of for loop for the information.

end % end of the Noise.

% Datagaps.
if strcmp(statanalysis, stattype6)

% put the total number of Datagaps into a single array.
total_all = [total_datagap];

% print useful information into the file.
fprintf(fid_dbase, '\n\n\nData Base Search Results for Data
Gaps\n');
fprintf(fid_dbase, 'Year : %d\n', year);
fprintf(fid_dbase, 'Month: %d\n', currentmonth);

```

```

fprintf(fid_dbase,'Beam : Widebeam\n');
fprintf(fid_dbase, ...
        '\nTotal Number of Data Gaps : %d\n',sum(total_all));

% go through all the requested event.
for y = 2:total_all(:, currentmonth)+1

fprintf(fid_dbase,'\nData Gap Event no.%d', y-1);
fprintf(fid_dbase,'\nStart Time Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
        total_datagap_startday(y), ...
        total_datagap_startmonth(y), ...
        total_datagap_startyear(y), ...
        total_datagap_starthour(y), ...
        total_datagap_startmin(y), ...
        total_datagap_startsec(y));
fprintf(fid_dbase,'\nEnd Time   Date %.2d/%.2d/%.2d Time
%.2d:%.2d:%.2d', ...
        total_datagap_endday(y), ...
        total_datagap_endmonth(y), ...
        total_datagap_endyear(y), ...
        total_datagap_endhour(y), ...
        total_datagap_endmin(y), ...
        total_datagap_endsec(y));
fprintf(fid_dbase,'\nDuration   %ddays and %.2dh %.2dm %.2ds',
...
        total_datagap_durday(y), ...
        total_datagap_durhour(y), ...
        total_datagap_durmin(y), ...
        total_datagap_dursec(y));
fprintf(fid_dbase,'\nMax           %.2fdB', ...
        total_datagap_max(y));
fprintf(fid_dbase,'\nMean           %.2fdB\n', ...
        total_datagap_mean(y));

end % end of for loop for the information.

```

```
end % end of the Datagaps.

% if error occur catch the ext commands.
catch

    % print a message on the Matlab Workspace.
    fprintf('There are No results available for the Month %d
(Year %d)\n', currentmonth, year);

    % if the current month failed to load (i.e. does not
exists), load the next one.
    currentmonth = currentmonth + 1;

end % end of try catch statement.

end % end of for loop for the months.

% close the file that has been opened for the
% results of the dbase analysis.
status = fclose(fid_dbase);
```

showevent.m

```
function showevent(yearst, monthst, dayst, hourst, minst,
eventplottype)

%SHOUEVENT
%
%Date Started: 17/07/2000 8:42pm
%Date Completed: 18/07/2000 12:28am
%Creator: G.Nikitas
%DCS, Lancaster University
%
% plots an event, that has been requested by the user.
% usually used, after the dbaseiris.m function.
% once we get the results from the above function, then
% we may want to view a specific event, so the solution is
% to call the showeventfunction.
% it searches the catalogues with the results then finds
% and collects information for the requested event;
% finally loads the iris data for the specific event and
% plots it with all the useful information
% (start and end time, max and mean value, duration etc.)
%
% the inputs the showevent accepts are,
% information of the start date and time the requested event
% occurred.
% i.e. the year started    : 1994
%       the month started  : 11
%       the day started    : 01
%       the hour started   : 02
%       the minute started : 13
% e.g. showevent(1994, 11, 01, 02, 13).

% set the default event for the statistical analysis.
if nargin < 6
    evetplottype = 'advanced';
```

```
end

% two different cases for the event to plot.
plottype1 = 'simple';
plottype2 = 'advanced';

% check if the correct input for the plot type has been
entered.
if ~strcmp(eventplottype, plottype1) & ~strcmp(eventplottype,
plottype2)

    % print a message on the workspace.
    fprintf('The input %s is not valid.\n', eventplottype);
    fprintf('valid input: simple or advanced\n');

    % exit the function and return to the matlab workspace.
    return;

end % end of if statement for the event plot type input.

% display a message on the Matlab Workspace.
fprintf('\nLoading Requested Event. Please Wait!\n');

% initialise some various variables.
eventtype = 0;
eventmax = 0;
eventmean = 0;
yearet = 0;
monthet = 0;
dayet = 0;
houret = 0;
minet = 0;
durday = 0;
durhour = 0;
durmin = 0;
```

```
dursec      = 0;

% if the month you entered is between 1 and 11
% then execute this if statement.
if ((monthst == 1 | monthst == 2 | monthst == 3 | monthst == 4
    | ...
    monthst == 5 | monthst == 6 | monthst == 7 | monthst ==
8 | ...
    monthst == 9 | monthst == 10 | monthst == 11) ...
    & (yearst >= 1994))

% specify the general characteristics of the results files.
nextyear = yearst + 1;
nextmonth = monthst + 1;
results_starttime = time([yearst monthst 01 0 0 0]);
results_endtime = time([yearst nextmonth 01 0 0 0]);

% the name of the files for the results depends on the
% requested year and month, that have been given by the user.
resultsfile = ...
    char(['zfilter' ...
        strftime(results_starttime,'%Y') ...
        strftime(results_starttime,'%m') '01_' ...
        strftime(results_endtime,'%Y') ...
        strftime(results_endtime,'%m') ...
        '01.gn']);

% if the month you entered is between 12
% then execute this elseif statement.
elseif (monthst == 12 & yearst >= 1994)

% specify the general characteristics of the results files.
nextyear = yearst + 1;
nextmonth = monthst + 1;
results_starttime = time([yearst monthst 01 0 0 0]);
results_endtime = time([nextyear 01 01 0 0 0]);
```



```
% the name of the files for the results depends on the
% requested year and month, that have been given by the user.
resultsfile = ...
    char(['zfilter' ...
        strftime(results_starttime,'%Y') ...
        strftime(results_starttime,'%m') '01_' ...
        strftime(results_endtime,'%Y') ...
        strftime(results_endtime,'%m') ...
        '01.gn']);

end % end of if-else statement.

% load the all files contains the results of the data analysis
for
% the requested year.
results = load([resultsfile]);

% find the size of the results matrix.
results_size = size(results);

% initialise a counter for the while loop below.
results_counter = 1;

% start from the begin of the analysis results until
% the last result.
while results_counter <= (results_size(1))

    % if the following statement is true, the requested event is
    found.
    if (results(results_counter, 3) == yearst) & ...
        (results(results_counter, 4) == monthst) & ...
        (results(results_counter, 5) == dayst) & ...
        (results(results_counter, 6) == hourst) & ...
        (results(results_counter, 7) == minst)
```

```
% collect useful information for the requested event.
eventtype = results(results_counter, 1);
eventmax  = results(results_counter, 19);
eventmean = results(results_counter, 20);
yearet    = results(results_counter, 9);
monthet   = results(results_counter, 10);
dayet     = results(results_counter, 11);
houret    = results(results_counter, 12);
minet     = results(results_counter, 13);
durday    = results(results_counter, 15);
durhour   = results(results_counter, 16);
durmin    = results(results_counter, 17);
dursec    = results(results_counter, 18);

end % end if if-else statement for the requested event.

% increament the counter by one.
results_counter = results_counter + 1;

end % end of while loop for the results search.

% check if the requested event does exists.
if eventtype == 0

    % if message not found display a message on the Matlab
    workspace.
    fprintf('\nWarning! Requested Event <%.2d/%.2d/%.2d
    %.2d:%.2d:00> Never Occured.\n', dayst, monthst, yearst,
    hourst, minst);

    % exit the function and return to Matlab workspace.
    return;
```

```
end % end of if statement that checks for the existence of the
    % requested event.

% plot the event with the simple type.
if strcmp(eventplotttype, plotttype1)

% call the function that loads the requested data.
% it returns useful information about the loaded data.
iabs = irisabs(time([yearst monthst dayst hourst minst 0]),
...
    time([yearet montheet dayet houret minet 0]), ...
    timespan(1, 'm'), kil, [50]);

% get the values (dBs) of the requested event.
eventvalues = getabs(iabs);

% find the size of eventvalues.
eventvalues_size = size(eventvalues);

% create a figure, to plot the event.
h_eventplot = figure;
plot(1:eventvalues_size(:,2), ...
    eventvalues(eventvalues_size(:,1), :), ...
    1:eventvalues_size(:,2), ...
    eventmax, ...
    1:eventvalues_size(:,2), ...
    eventmean);

ylabel('Absorption Level in dBs');
xlabel('Duration of Event in Minutes');
legend('Event', 'Max', 'Mean');
set(h_eventplot, ...
    'name', 'Kilpisjarvi, Finland. Widebeam, Resolution 1
min.');
```

```

% these will be used for the title string.
eventst = time([yearst monthst dayst hourst minst 00]);
eventet = time([yearet monthet dayet houret minet 00]);

% check the type of the requested event.
if eventtype == 2
    str1 = 'Data Gap Event';
    str2 = char([ ...
        'Started at ' ...
        strftime(eventst,'%H') ':' ...
        strftime(eventst,'%M') ':' ...
        strftime(eventst,'%S') ...
        ' ' ...
        strftime(eventst,'%d') '/' ...
        strftime(eventst,'%m') '/' ...
        strftime(eventst,'%Y') ...
        ' & ' ...
        'Finished at ' ...
        strftime(eventet,'%H') ':' ...
        strftime(eventet,'%M') ':' ...
        strftime(eventet,'%S') ...
        ' ' ...
        strftime(eventet,'%d') '/' ...
        strftime(eventet,'%m') '/' ...
        strftime(eventet,'%Y') ...
        '.']);
    titlestr = str2mat(str1, str2);
    title(titlestr);
end
if eventtype == 3
    str1 = 'Noise Event';
    str2 = char([ ...
        'Started at ' ...
        strftime(eventst,'%H') ':' ...
        strftime(eventst,'%M') ':' ...
        strftime(eventst,'%S') ...
        ' ' ...

```

```

        strftime(eventst, '%d') '/' ...
        strftime(eventst, '%m') '/' ...
        strftime(eventst, '%Y') ...
        ' & ' ...
        'Finished at ' ...
        strftime(eventet, '%H') ':' ...
        strftime(eventet, '%M') ':' ...
        strftime(eventet, '%S') ...
        ' ' ...
        strftime(eventet, '%d') '/' ...
        strftime(eventet, '%m') '/' ...
        strftime(eventet, '%Y') ...
        ' ']);
    titlestr = str2mat(str1, str2);
    title(titlestr);
end
if eventtype == 4
    str1 = 'Auroral Absorption (AA) Event';
    str2 = char([ ...
        'Started at ' ...
        strftime(eventst, '%H') ':' ...
        strftime(eventst, '%M') ':' ...
        strftime(eventst, '%S') ...
        ' ' ...
        strftime(eventst, '%d') '/' ...
        strftime(eventst, '%m') '/' ...
        strftime(eventst, '%Y') ...
        ' & ' ...
        'Finished at ' ...
        strftime(eventet, '%H') ':' ...
        strftime(eventet, '%M') ':' ...
        strftime(eventet, '%S') ...
        ' ' ...
        strftime(eventet, '%d') '/' ...
        strftime(eventet, '%m') '/' ...
        strftime(eventet, '%Y') ...
        ' ']);

```

```
titlestr = str2mat(str1, str2);
title(titlestr);
end
if eventtype == 5
    str1 = 'Polar Cap Absorption (PCA) Event';
    str2 = char([ ...
        'Started at ' ...
        strftime(eventst, '%H') ':' ...
        strftime(eventst, '%M') ':' ...
        strftime(eventst, '%S') ...
        ' ' ...
        strftime(eventst, '%d') '/' ...
        strftime(eventst, '%m') '/' ...
        strftime(eventst, '%Y') ...
        ' & ' ...
        'Finished at ' ...
        strftime(eventet, '%H') ':' ...
        strftime(eventet, '%M') ':' ...
        strftime(eventet, '%S') ...
        ' ' ...
        strftime(eventet, '%d') '/' ...
        strftime(eventet, '%m') '/' ...
        strftime(eventet, '%Y') ...
        '.']);
    titlestr = str2mat(str1, str2);
    title(titlestr);
end

end % end of if statement for the simple type of plot.

% plot the event with the advance type.
if strcmp(eventplotype, plotype2)

% call the function that loads the requested data.
% it returns useful information about the loaded data.
```

```
iabs = irisabs(time([yearst monthst dayst hourst minst 0]),
...
    time([yearet monthet dayet houret minet 0]), ...
    timespan(1, 'm'), kil, [1:50]);

% get the values (dBs) of the requested event.
eventvalues = getabs(iabs);

% find the size of eventvalues.
eventvalues_size = size(eventvalues);

% create a figure, to plot the event.
h_eventplot = figure;
plot(1:eventvalues_size(:,2), ...
    eventvalues(eventvalues_size(:,1), :), ...
    1:eventvalues_size(:,2), ...
    eventmax, ...
    1:eventvalues_size(:,2), ...
    eventmean);

ylabel('Absorption Level in dBs');
xlabel('Duration of Event in Minutes');
legend('Event', 'Max', 'Mean');
set(h_eventplot, ...
    'name', 'Kilpisjarvi, Finland. Widebeam, Resolution 1
min.');
```

```
% these will be used for the title string.
eventst = time([yearst monthst dayst hourst minst 00]);
eventet = time([yearet monthet dayet houret minet 00]);

% check the type of the requested event.
if eventtype == 2
    str1 = 'Data Gap Event';
    str2 = char([ ...
        'Started at ' ...
```

```

    strftime(eventst, '%H') ':' ...
    strftime(eventst, '%M') ':' ...
    strftime(eventst, '%S') ...
    ' ' ...
    strftime(eventst, '%d') '/' ...
    strftime(eventst, '%m') '/' ...
    strftime(eventst, '%Y') ...
    ' & ' ...
    'Finished at ' ...
    strftime(eventet, '%H') ':' ...
    strftime(eventet, '%M') ':' ...
    strftime(eventet, '%S') ...
    ' ' ...
    strftime(eventet, '%d') '/' ...
    strftime(eventet, '%m') '/' ...
    strftime(eventet, '%Y') ...
    '.']]);
titlestr = str2mat(str1, str2);
title(titlestr);
end
if eventtype == 3
    str1 = 'Noise Event';
    str2 = char([ ...
        'Started at ' ...
        strftime(eventst, '%H') ':' ...
        strftime(eventst, '%M') ':' ...
        strftime(eventst, '%S') ...
        ' ' ...
        strftime(eventst, '%d') '/' ...
        strftime(eventst, '%m') '/' ...
        strftime(eventst, '%Y') ...
        ' & ' ...
        'Finished at ' ...
        strftime(eventet, '%H') ':' ...
        strftime(eventet, '%M') ':' ...
        strftime(eventet, '%S') ...
        ' ' ...

```



```

        strftime(eventet, '%d') '/' ...
        strftime(eventet, '%m') '/' ...
        strftime(eventet, '%Y') ...
        '.']);
    titlestr = str2mat(str1, str2);
    title(titlestr);
end
if eventtype == 4
    str1 = 'Auroral Absorption (AA) Event';
    str2 = char([ ...
        'Started at ' ...
        strftime(eventst, '%H') ':' ...
        strftime(eventst, '%M') ':' ...
        strftime(eventst, '%S') ...
        ' ' ...
        strftime(eventst, '%d') '/' ...
        strftime(eventst, '%m') '/' ...
        strftime(eventst, '%Y') ...
        ' & ' ...
        'Finished at ' ...
        strftime(eventet, '%H') ':' ...
        strftime(eventet, '%M') ':' ...
        strftime(eventet, '%S') ...
        ' ' ...
        strftime(eventet, '%d') '/' ...
        strftime(eventet, '%m') '/' ...
        strftime(eventet, '%Y') ...
        '.']);
    titlestr = str2mat(str1, str2);
    title(titlestr);
end
if eventtype == 5
    str1 = 'Polar Cap Absorption (PCA) Event';
    str2 = char([ ...
        'Started at ' ...
        strftime(eventst, '%H') ':' ...
        strftime(eventst, '%M') ':' ...

```

```
    strftime(eventst, '%S') ...  
    ' ' ...  
    strftime(eventst, '%d') '/' ...  
    strftime(eventst, '%m') '/' ...  
    strftime(eventst, '%Y') ...  
    ' & ' ...  
    'Finished at ' ...  
    strftime(eventet, '%H') ':' ...  
    strftime(eventet, '%M') ':' ...  
    strftime(eventet, '%S') ...  
    ' ' ...  
    strftime(eventet, '%d') '/' ...  
    strftime(eventet, '%m') '/' ...  
    strftime(eventet, '%Y') ...  
    '.']);  
    titlestr = str2mat(str1, str2);  
    title(titlestr);  
end  
  
% plot the event on contours as well.  
imageplot(iabs);  
  
end % end of if statement for the advance type of plot.
```

stataa_tod.m

```
function stataa_tod(year, months, beams);

%STATAA_TOD
%
%Date Started: 21/06/2000 1:57am
%Date Completed: 22/06/2000 2:55am
%Creator: G.Nikitas
%DCS, Lancaster University
%
% the input arguments the function takes are the year, the
months
% and which beams the results file contains.
% therefore the input beams accepts the value:
% 'widebeam' for the wide beam and 'allbeams' for all the 50
% beams. if the input beam is not been given
% a default value is set to the 'widebeam'.
% the program then load the file which contains the results of
the
% data analysis for this period of time.
% next finds the number of Auroral Absorptions that have been
% occurred in that time; split the AAs into 6 differet groups
% depending on the time of occurrence.
% Group1, AAs / Spikes with time of occurrece: 00:00:00 to
04:00:00
% Group2, AAs / Spikes with time of occurrece: 04:00:00 to
08:00:00
% Group3, AAs / Spikes with time of occurrece: 08:00:00 to
12:00:00
% Group4, AAs / Spikes with time of occurrece: 12:00:00 to
16:00:00
% Group5, AAs / Spikes with time of occurrece: 16:00:00 to
20:00:00
% Group6, AAs / Spikes with time of occurrece: 20:00:00 to
24:00:00
% next the program plots a histogram showing the number of AAs
```

```
for
% each group for the requested time (e.g. for the month 05
year
% 1998).

% set the default value for the beams to the widebeam.
if nargin < 3
    beams = 'widebeam';
end

% set the default value for the months.
if nargin < 2
    months = [1:12];
end

% two different cases for the beams input.
beamstype1 = 'widebeam';
beamstype2 = 'allbeams';

% check if the correct input for the beams has been entered.
if ~strcmp(beams, beamstype1) & ~strcmp(beams, beamstype2)

    % print a message on the workspace.
    fprintf('The input %s is not valid.\n', beams);
    fprintf('valid input: widebeam or allbeams\n');

    % exit the function and return to the matlab workspace.
    return;

end % end of if statement for the beams type input.

% take one month each time and apply the statistical analysis.
for month = months
```

```
% try to load and analyse the following results file, if
failed to
% to load any file go to catch (try to load the ext file).
try

% depending on the inputs given by the user,
% initialise these two variables.
next_month = month + 1;
next_year = year + 1;

% if the month you entered is between 1 and 11
% then execute this if statement.
if ((month == 1 | month == 2 | month == 3 | month == 4 | ...
    month == 5 | month == 6 | month == 7 | month == 8 | ...
    month == 9 | month == 10 | month == 11) ...
    & (year >= 1994))

% specify the general characteristics of the results file.
results_starttime = ...
    time([year month 01 0 0 0]);      % start date - to load
the results.
results_endtime = ...
    time([year next_month 01 0 0 0]); % end date - to load
the results.

% if the month you entered is 12
% then execute this elseif statement.
elseif (month == 12 & year >= 1994)

% specify the general characteristics of the results file.
results_starttime = ...
    time([year month 01 0 0 0]);      % start date - to load the
results.
results_endtime = ...
    time([next_year 01 01 0 0 0]);    % end date - to load ther
esults.
```

```
% if the month you entered is not between 1 to 11 or 12
% then execute this statement.
else

    fprintf('Invalid entry for the month (%d) or for the year
    (%d)\n', month, year);
    return;

end % end of if-elseif-else statement.


% the name of the file for the results depends on the
% requested start time, that have been given by the user.
analysisresults = ...
    char(['zfilter' strftime(results_starttime,'%Y%m') '01_'
    ...
    strftime(results_endtime,'%Y%m') '01.gn']);

% load the results of the analysis.
results = load([analysisresults]);

% find the size of the results matrix.
results_size = size(results);


% initialise the counters that keep track of the number
% of Auroral Absorptions that occur in that period of time.
aa_counter_group1 = 0;
aa_counter_group2 = 0;
aa_counter_group3 = 0;
aa_counter_group4 = 0;
aa_counter_group5 = 0;
aa_counter_group6 = 0;


% initialise the counters that keep track of the number
% of Auroral Absorptions that occur in that period of time,
% BUT whos duration is less than 6 minutes (i.e. for AA which
```

```
% look like spkes).
aal_counter_group1 = 0;
aal_counter_group2 = 0;
aal_counter_group3 = 0;
aal_counter_group4 = 0;
aal_counter_group5 = 0;
aal_counter_group6 = 0;

% initialise a counter for the while loop below.
results_counter = 1;

% start from the begin of the analysis results until
% the last result.
while results_counter <= (results_size(1))

    % study the number of auroral absorptions
    % with duration from 2 mins to 5 hours.
    % i.e. find ALL the AA events.

    % group 1.
    % find the number of AA events whos time of occurrence
    % was between 00:00:00 and 04:00:00.
    if (results(results_counter, 1) == 4) & ...
        (results(results_counter, 6) >= 0 & ...
        results(results_counter, 6) <= 3)

        % the if statement is true, increament the counter by one.
        aa_counter_group1 = aa_counter_group1 + 1;

    end % end of if statement that looks for the group 1 of AA
    events.

    % group 2.
    % find the number of AA events whos time of occurrence
    % was between 04:00:00 and 08:00:00.
    if (results(results_counter, 1) == 4) & ...
```

```
(results(results_counter, 6) >= 4 & ...
results(results_counter, 6) <= 7)

% the if statement is true, increament the counter by one.
aa_counter_group2 = aa_counter_group2 + 1;

end % end of if statement that looks for the group 2 of AA
events.

% group 3.
% find the number of AA events whos time of occurrence
% was between 08:00:00 and 12:00:00.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 6) >= 8 & ...
    results(results_counter, 6) <= 11)

    % the if statement is true, increament the counter by one.
    aa_counter_group3 = aa_counter_group3 + 1;

end % end of if statement that looks for the group 3 of AA
events.

% group 4.
% find the number of AA events whos time of occurrence
% was between 12:00:00 and 16:00:00.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 6) >= 12 & ...
    results(results_counter, 6) <= 15)

    % the if statement is true, increament the counter by one.
    aa_counter_group4 = aa_counter_group4 + 1;

end % end of if statement that looks for the group 4 of AA
events.

% group 5.
% find the number of AA events whos time of occurrence
```



```
% was between 16:00:00 and 20:00:00.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 6) >= 16 & ...
    results(results_counter, 6) <= 19)

    % the if statement is true, increament the counter by one.
    aa_counter_group5 = aa_counter_group5 + 1;

end % end of if statement that looks for the group 5 of AA
events.

% group 6.
% find the number of AA events whos time of occurrence
% was between 20:00:00 and 24:00:00.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 6) >= 20 & ...
    results(results_counter, 6) <= 23)

    % the if statement is true, increament the counter by one.
    aa_counter_group6 = aa_counter_group6 + 1;

end % end of if statement that looks for the group 6 of AA
events.

% study the number of auroral absorptions
% with duration from 2 mins to 6 minutes.
% i.e. find the SPIKES AA events.

% group 1.
% find the number of AA events whos time of occurrence
% was between 00:00:00 and 04:00:00.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 16) == 0) & ...
    (results(results_counter, 17) <= 6) & ...
    (results(results_counter, 6) >= 0 & ...
    results(results_counter, 6) <= 3)
```

```
% the if statement is true, increament the counter by one.
aal_counter_group1 = aal_counter_group1 + 1;

end % end of if statement that looks for the group 1 of AA
events.

% group 2.
% find the number of AA events whos time of occurrence
% was between 04:00:00 and 08:00:00.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 16) == 0) & ...
    (results(results_counter, 17) <= 6) & ...
    (results(results_counter, 6) >= 4 & ...
    results(results_counter, 6) <= 7)

    % the if statement is true, increament the counter by one.
    aal_counter_group2 = aal_counter_group2 + 1;

end % end of if statement that looks for the group 2 of AA
events.

% group 3.
% find the number of AA events whos time of occurrence
% was between 08:00:00 and 12:00:00.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 16) == 0) & ...
    (results(results_counter, 17) <= 6) & ...
    (results(results_counter, 6) >= 8 & ...
    results(results_counter, 6) <= 11)

    % the if statement is true, increament the counter by one.
    aal_counter_group3 = aal_counter_group3 + 1;

end % end of if statement that looks for the group 3 of AA
events.
```

```
% group 4.
% find the number of AA events whos time of occurrence
% was between 12:00:00 and 16:00:00.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 16) == 0) & ...
    (results(results_counter, 17) <= 6) & ...
    (results(results_counter, 6) >= 12 & ...
    results(results_counter, 6) <= 15)

    % the if statement is true, increament the counter by one.
    aal_counter_group4 = aal_counter_group4 + 1;

end % end of if statement that looks for the group 4 of AA
events.

% group 5.
% find the number of AA events whos time of occurrence
% was between 16:00:00 and 20:00:00.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 16) == 0) & ...
    (results(results_counter, 17) <= 6) & ...
    (results(results_counter, 6) >= 16 & ...
    results(results_counter, 6) <= 19)

    % the if statement is true, increament the counter by one.
    aal_counter_group5 = aal_counter_group5 + 1;

end % end of if statement that looks for the group 5 of AA
events.

% group 6.
% find the number of AA events whos time of occurrence
% was between 20:00:00 and 24:00:00.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 16) == 0) & ...
    (results(results_counter, 17) <= 6) & ...
    (results(results_counter, 6) >= 20 & ...
```

```
results(results_counter, 6) <= 23)

% the if statement is true, increament the counter by one.
aal_counter_group6 = aal_counter_group6 + 1;

end % end of if statement that looks for the group 6 of AA
events.

% increament the counter by one, to read the next event from
the
% results matrix.
results_counter = results_counter + 1;

end % end of while loop used to read the analysis results.

% if the results file contains the analysis of the wide beam
only.
if strcmp(beams, beamstype1)

% collect the total number of AAs for each group.
aa_total_group1 = aa_counter_group1;
aa_total_group2 = aa_counter_group2;
aa_total_group3 = aa_counter_group3;
aa_total_group4 = aa_counter_group4;
aa_total_group5 = aa_counter_group5;
aa_total_group6 = aa_counter_group6;

% collect the total number of SPIKES AAs for each group.
aal_total_group1 = aal_counter_group1;
aal_total_group2 = aal_counter_group2;
aal_total_group3 = aal_counter_group3;
aal_total_group4 = aal_counter_group4;
aal_total_group5 = aal_counter_group5;
aal_total_group6 = aal_counter_group6;
```

```
end % end of if statement for the widebeam.

% if the results file contains the analysis of all the beams.
if strcmp(beams, beamstype2)

% collect the total number of AAs for each group.
% apply the averaging factor.
aa_total_group1 = aa_counter_group1 / 50;
aa_total_group2 = aa_counter_group2 / 50;
aa_total_group3 = aa_counter_group3 / 50;
aa_total_group4 = aa_counter_group4 / 50;
aa_total_group5 = aa_counter_group5 / 50;
aa_total_group6 = aa_counter_group6 / 50;

% collect the total number of SPIKES AAs for each group.
% apply the averaging factor.
aal_total_group1 = aal_counter_group1 / 50;
aal_total_group2 = aal_counter_group2 / 50;
aal_total_group3 = aal_counter_group3 / 50;
aal_total_group4 = aal_counter_group4 / 50;
aal_total_group5 = aal_counter_group5 / 50;
aal_total_group6 = aal_counter_group6 / 50;

end % end of if statement for all the beams.

% print a message on the workspace.
fprintf('\nStatistical Analysis\nYear %d, Month %d\n', year,
month);
fprintf('<Group 1> Time of Occurrence for AA / Spike: 00:00:00
and 04:00:00\n');
fprintf('<Group 2> Time of Occurrence for AA / Spike: 04:00:00
and 08:00:00\n');
fprintf('<Group 3> Time of Occurrence for AA / Spike: 08:00:00
and 12:00:00\n');
fprintf('<Group 4> Time of Occurrence for AA / Spike: 12:00:00
and 16:00:00\n');
```

```

fprintf('<Group 5> Time of Occurrence for AA / Spike: 16:00:00
and 20:00:00\n');
fprintf('<Group 6> Time of Occurrence for AA / Spike: 20:00:00
and 24:00:00\n');
fprintf('\n');
fprintf('<Group 1> Number of all AAs / Spikes: %.1f / %.1f\n',
...
    aa_total_group1, aal_total_group1);
fprintf('<Group 2> Number of all AAs / Spikes: %.1f / %.1f\n',
...
    aa_total_group2, aal_total_group2);
fprintf('<Group 3> Number of all AAs / Spikes: %.1f / %.1f\n',
...
    aa_total_group3, aal_total_group3);
fprintf('<Group 4> Number of all AAs / Spikes: %.1f / %.1f\n',
...
    aa_total_group4, aal_total_group4);
fprintf('<Group 5> Number of all AAs / Spikes: %.1f / %.1f\n',
...
    aa_total_group5, aal_total_group5);
fprintf('<Group 6> Number of all AAs / Spikes: %.1f / %.1f\n',
...
    aa_total_group6, aal_total_group6);

% put the total number of all AAs of each group into an array.
aa_total = [aa_total_group1 ...
    aa_total_group2 ...
    aa_total_group3 ...
    aa_total_group4 ...
    aa_total_group5 ...
    aa_total_group6];

% put the total number of Spikes AAs of each group into an
array.
aal_total = [aal_total_group1 ...
    aal_total_group2 ...

```

```
aal_total_group3 ...
aal_total_group4 ...
aal_total_group5 ...
aal_total_group6];

% put the total number of all AAs and the total number of
Spikes,
% into a single array.
aa_all_and_spikes = [aa_total; aal_total];

% open a new figure for the histogram.
h_aatod = figure;

% plot the number of all AAs and Spike AAs
% of each group in a histogram.
% 1 bar from the histogram represents
% the number of AAs of 1 single group.
bar(aa_all_and_spikes', 'grouped');

% add useful information on the histogram.
titlestr = ...
    char(['Auroral Absorptions & Spikes. (Month '...
        strftime(results_starttime,'%m') ', Year ' ...
        strftime(results_starttime,'%Y') ')]);
title(titlestr);
xlabel(['6 Different Groups (each group represents a different
time of the' ...
    ' day)']);
ylabel('Number of Events');
legend('ALL AAs', 'Spikes');
grid
set(h_aatod, 'name', 'Statistical Analysis I');

% if error occur catch the next commands.
catch
```

```
% print a message on the Matlab Workspace.  
fprintf('\nThere are No results available for the Month %d  
(Year %d)\n', month, year);  
  
end % end of try-catch statement.  
  
end % end of for loop for the months.
```


stataa_dur.m

```
function stataa_dur(year, months, beams);

%STATAA_DUR
%
%Date Started: 20/06/2000 8:57pm
%Date Completed: 22/06/2000 3:05am
%Creator: G.Nikitas
%DCS, Lancaster University
%
% the user specifies the start time for the year, the months
and
% the beams that have been analysed into the results file.
% the program then load the file which contains the results of
the
% data analysis for this period of time.
% next finds the number of Auroral Absorptions that have been
% occurred in that time; split the AAs into 6 differet groups
% depending on their duration of occurrence.
% Group1, AAs and Spikes with duration: > 02 mins < 06 mins
% Group2, AAs and Spikes with duration: > 06 mins < 30 mins
% Group3, AAs and Spikes with duration: > 30 mins < 01 hours
% Group4, AAs and Spikes with duration: > 01 hours < 02 hours
% Group5, AAs and Spikes with duration: > 02 hours < 03 hours
% Group6, AAs and Spikes with duration: > 03 hours < 05 hours
% next the program plots a histogram showing the number of AAs
for
% each group for the requested time (e.g. for the month 05
year
% 1998) .
% mote that the input beams accepts the value:
% 'widebeam' for the wide beam and 'allbeams' for all the 50
% beams if the input beam is not been given
% a default value is set to the 'widebeam'.
```

```
% set the default value for the beams to the widebeam.
if nargin < 3
    beams = 'widebeam';
end

% set the default value for the months.
if nargin < 2
    months = [1:12];
end

% two different cases for the beams input.
beamstype1 = 'widebeam';
beamstype2 = 'allbeams';

% check if the correct input for the beams has been entered.
if ~strcmp(beams, beamstype1) & ~strcmp(beams, beamstype2)

    % print a message on the workspace.
    fprintf('The input %s is not valid.\n', beams);
    fprintf('valid inputs: widebeam or allbeams\n');

    % exit the function and return to the matlab workspace.
    return;

end % end of if statement for the beams type input.

% take one month each time and apply the statistical analysis.
for month = months

    % try to load and analyse the following results file, if
    failed to
    % to load any file go to catch (try to load the ext file).
    try
```

```
% depending on the inputs given by the user,
% initialise these two variables.
next_month = month + 1;
next_year = year + 1;

% if the month you entered is between 1 and 11
% then execute this if statement.
if ((month == 1 | month == 2 | month == 3 | month == 4 | ...
    month == 5 | month == 6 | month == 7 | month == 8 | ...
    month == 9 | month == 10 | month == 11) ...
    & (year >= 1994))

% specify the general characteristics of the results file.
results_starttime = ...
    time([year month 01 0 0 0]);      % start date - to load
the results.
results_endtime = ...
    time([year next_month 01 0 0 0]); % end date - to load
the results.

% if the month you entered is 12
% then execute this elseif statement.
elseif (month == 12 & year >= 1994)

% specify the general characteristics of the results file.
results_starttime = ...
    time([year month 01 0 0 0]);      % start date - to load the
results.
results_endtime = ...
    time([next_year 01 01 0 0 0]);    % end date - to load the
results.

% if the month you entered is not between 1 to 11 or 12
% then execute this statement.
else
```

```
fprintf('Invalid entry for the month (%d) or for the year
(%d)\n', month, year);
return;

end % end of if-elseif-else statement.

% the name of the file for the results depends on the
% requested start time, that have been given by the user.
analysisresults = ...
    char(['zfilter' strftime(results_starttime,'%Y%m') '01_'
...
    strftime(results_endtime,'%Y%m') '01.gn']);

% load the results of the analysis.
results = load([analysisresults]);

% find the size of the results matrix.
results_size = size(results);

% initialise the counters that keep track of the number
% of Auroral Absorptions that occur in that period of time.
aa_counter_group1 = 0;
aa_counter_group2 = 0;
aa_counter_group3 = 0;
aa_counter_group4 = 0;
aa_counter_group5 = 0;
aa_counter_group6 = 0;

% initialise a counter for the while loop below.
results_counter = 1;

% start from the begin of the analysis results until
% the last result.
while results_counter <= (results_size(1))

    % group 1.
```

```
% find the number of AA events whos duration was
% between 2 to 6 minutes.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 16) == 0) & ...
    (results(results_counter, 17) <= 6)

    % the if statement is true, increament the counter by one.
    aa_counter_group1 = aa_counter_group1 + 1;

end % end of if statement that looks for the group 1 of AA
events.

% group 2.
% find the number of AA events whos duration was
% between 6 to 30 minutes.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 16) == 0) & ...
    (results(results_counter, 17) > 6 & ...
    results(results_counter, 17) <= 30)

    % the if statement is true, increament the counter by one.
    aa_counter_group2 = aa_counter_group2 + 1;

end % end of if statement that looks for the group 2 of AA
events.

% group 3.
% find the number of AA events whos duration was
% between 30 minutes to < 1 hour.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter, 16) == 0) & ...
    (results(results_counter, 17) > 30 & ...
    results(results_counter, 17) <= 59)

    % the if statement is true, increament the counter by one.
    aa_counter_group3 = aa_counter_group3 + 1;
```

```
end % end of if statement that looks for the group 3 of AA
events.

% group 4.
% find the number of AA events whos duration was
% between 1 to < 2 hours.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter,16) == 1)

    % the if statement is true, increament the counter by one.
    aa_counter_group4 = aa_counter_group4 + 1;

end % end of if statement that looks for the group 4 of AA
events.

% group 5.
% find the number of AA events whos duration was
% between 2 to 3 hours.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter,16) == 2)

    % the if statement is true, increament the counter by one.
    aa_counter_group5 = aa_counter_group5 + 1;

end % end of if statement that looks for the group 5 of AA
events.

% group 6.
% find the number of AA events whos duration was
% between 3 to < 5 hours (the max duration for AA is 5
hours).
% (Note that an event can last e.g. 4 hours and 50 minutes.
if (results(results_counter, 1) == 4) & ...
    (results(results_counter,16) >= 30)

    % the if statement is true, increament the counter by one.
    aa_counter_group6 = aa_counter_group6 + 1;
```

```
    end % end of if statement that looks for the group 6 of AA
    events.

    % increament the counter by one, to read the next event from
    the
    % results matrix.
    results_counter = results_counter + 1;

end % end of while loop used to read the analysis results.

% if the results file contains the analysis of the widebeam.
if strcmp(beams, beamstype1)

% collect the total number of AAs for each group.
aa_total_group1 = aa_counter_group1;
aa_total_group2 = aa_counter_group2;
aa_total_group3 = aa_counter_group3;
aa_total_group4 = aa_counter_group4;
aa_total_group5 = aa_counter_group5;
aa_total_group6 = aa_counter_group6;

end % end of if statement for the widebeam.

% if the results file contains the analysis of all the beams.
if strcmp(beams, beamstype2)

% collect the total number of AAs for each group.
% apply the averaging factor.
aa_total_group1 = aa_counter_group1 / 50;
aa_total_group2 = aa_counter_group2 / 50;
aa_total_group3 = aa_counter_group3 / 50;
aa_total_group4 = aa_counter_group4 / 50;
aa_total_group5 = aa_counter_group5 / 50;
aa_total_group6 = aa_counter_group6 / 50;

end % end of if statement for all the beams.
```

```
% print a message on the workspace.
fprintf('\nStatistical Analysis\nYear %d, Month %d\n', year,
month);
fprintf('<Group 1> Duration of AA (Spikes): > 02 mins < 06
mins\n');
fprintf('<Group 2> Duration of AA: > 06 mins < 30 mins\n');
fprintf('<Group 3> Duration of AA: > 30 mins < 01 hours\n');
fprintf('<Group 4> Duration of AA: > 01 hours < 02 hours\n');
fprintf('<Group 5> Duration of AA: > 02 hours < 03 hours\n');
fprintf('<Group 6> Duration of AA: > 03 hours < 05 hours\n');
fprintf('\n');
fprintf('<Group 1> Number of AAs (Spikes): %.1f\n',
aa_total_group1);
fprintf('<Group 2> Number of AAs : %.1f\n', aa_total_group2);
fprintf('<Group 3> Number of AAs : %.1f\n', aa_total_group3);
fprintf('<Group 4> Number of AAs : %.1f\n', aa_total_group4);
fprintf('<Group 5> Number of AAs : %.1f\n', aa_total_group5);
fprintf('<Group 6> Number of AAs : %.1f\n', aa_total_group6);

% put the total number of AAs of each group into an array.
aa_total = [aa_total_group1 ...
    aa_total_group2 ...
    aa_total_group3 ...
    aa_total_group4 ...
    aa_total_group5 ...
    aa_total_group6];

% open a new figure for the histogram.
h_aadur = figure;

% plot the number of AAs of each group in a histogram.
% 1 bar from the histogram represents
% the number of AAs of 1 single group.
bar(1:length(aa_total), aa_total);

% add useful information on the histogram.
```



```
titlestr = ...
    char(['Auroral Absorptions & Spikes. (Month '...
        strftime(results_starttime,'%m') ', Year ' ...
        strftime(results_starttime,'%Y') ')']);
title(titlestr);
xlabel('6 Different Groups of Durations');
ylabel('Number of Events');
grid
set(h_aadur, 'name', 'Statistical Analysis II');

% if error occur catch the next commands.
catch

    % print a message on the Matlab Workspace.
    fprintf('\nThere are No results available for the Month %d
(Year %d)\n', month, year);

end % end of try-catch statement.

end % end of for loop for the months.
```

statmultilong.m

```
function statmultilong(year, loadmonths, statanalysis, beams)

%STATMULTILONG
%
%Date Started: 20/06/2000 8:57pm
%Date Completed: 22/06/2000 3:05am
%Creator: G.Nikitas
%DCS, Lancaster University
%
% the program loads the contents of the results file depending
on
% the user's inputs.
% next it plots the results of the statistical analysis on a
% histogram; depending on the user's inputs.
%
% the user should specify the following inputs:
% year - for the year he wants to apply the analysis. e.g.
1995
% loadmonths - for the months he is interested. e.g. [4 7
9:11]
% statanalysis - for the type of analysis he wants to apply.
% e.g. 'all', 'spike', 'aa', 'pca', 'noise' or 'datagap'
% beams - on which beams he wants to apply the analysis.
% e.g. 'widebeam' or 'allbeams'

% set the default value for the beams.
if nargin < 4
    beams = 'widebeam';
end

% set the default event for the statistical analysis.
if nargin < 3
    statanalysis = 'all';
end
```

```
% set the default value for the loadmonths.
if nargin < 2
    loadmonths = [1:12];
end

% six different cases for the type of the statistical
analysis.
stattype1 = 'all';
stattype2 = 'spike';
stattype3 = 'aa';
stattype4 = 'pca';
stattype5 = 'noise';
stattype6 = 'datagap';

% two different cases for the beams input.
beamstype1 = 'widebeam';
beamstype2 = 'allbeams';

% check if the correct input for the statistical analysis has
been entered.
if ~strcmp(statanalysis, stattype1) & ...
    ~strcmp(statanalysis, stattype2) & ...
    ~strcmp(statanalysis, stattype3) & ...
    ~strcmp(statanalysis, stattype4) & ...
    ~strcmp(statanalysis, stattype5) & ...
    ~strcmp(statanalysis, stattype6)

    % print a message on the workspace.
    fprintf('The input %s is not valid.\n', statanalysis);
    fprintf('valid inputs: all, spike, aa, pca, noise,
datagap\n');

    % exit the function and return to the matlab workspace.
    return;
```

```
end % end of if statement for the input of the statistical
analysis.

% check if the correct input for the beams has been entered.
if ~strcmp(beams, beamstype1) & ~strcmp(beams, beamstype2)

    % print a message on the workspace.
    fprintf('The input %s is not valid.\n', beams);
    fprintf('valid input: widebeam or allbeams\n');

    % exit the function and return to the matlab workspace.
    return;

end % end of if statement for the beams type input.

% try to load and analyse the following results file, if
failed to
% to load the file for the current year;
% go to the catch statement.
try

% specify the general characteristics of the results files.
nextyear = year + 1;
results_starttime = time([year 01 01 0 0 0]);
results_endtime = time([nextyear 01 01 0 0 0]);

% the name of the files for the results depends on the
% requested year, that have been given by the user.
resultsfile = ...
    char(['zcatalogue' strftime(results_starttime,'%Y')
'.gn']);

% load the all files contains the results of the data analysis
for
% the requested year.
```

```
results = load([resultsfile]);

% find the size of the results matrix.
results_size = size(results);

% initialise the counters that keep track of the number
% of different events that occur in one month.
counter_aa_spikes = 0;
counter_noise = 0;
counter_aa_all = 0;
counter_noise = 0;
counter_datagap = 0;
counter_pca = 0;
counter_noise = 0;
counter_datagap = 0;

% initialise the variables that keep the total number
% of different events that occur every month for one year.
total_pca = zeros(1, 12);
total_noise = zeros(1, 12);
total_datagap = zeros(1, 12);
total_aa_spikes = zeros(1, 12);
total_aa_all = zeros(1, 12);
total_pca = zeros(1, 12);
total_noise = zeros(1, 12);
total_datagap = zeros(1, 12);

% we want to find the number of different events
% that occur in one month and repeat this for one year.
% so start from the first month of the year until the last
% one.
for currentmonth = loadmonths

% initialise a counter for the while loop below.
```

```
results_counter = 1;

% start from the begin of the analysis results until
% the last result.
while results_counter <= (results_size(1))

    % find the SPIKES - AURORAL ABSORPTIONS.
    if (results(results_counter, 1) == 4) & ...
        (results(results_counter, 16) == 0) & ...
        (results(results_counter, 17) <= 6) & ...
        (results(results_counter, 4) == currentmonth)

        % the if statement is true, increament the counter by one.
        counter_aa_spikes = counter_aa_spikes + 1;

    end % end of if statement that looks for the SPIKES - AAs.

    % find ALL the AURORAL ABSORPTIONS.
    if (results(results_counter, 1) == 4) & ...
        (results(results_counter, 4) == currentmonth)

        % the if statement is true, increament the counter by one.
        counter_aa_all = counter_aa_all + 1;

    end % end of if statement that looks for the ALL - AAs.

    % find the POLAR CAP ABSORPTIONS.
    if (results(results_counter, 1) == 5) & ...
        (results(results_counter, 4) == currentmonth)

        % the if statement is true, increament the counter by one.
        counter_pca = counter_pca + 1;

    end % end of if statement that looks for the PCAs.
```

```
% find the NOISE.
if (results(results_counter, 1) == 3) & ...
    (results(results_counter, 4) == currentmonth)

    % the if statement is true, increament the counter by one.
    counter_noise = counter_noise + 1;

end % end of if statement that looks for the NOISE.

% find the DATAGAPS.
if (results(results_counter, 1) == 2) & ...
    (results(results_counter, 4) == currentmonth)

    % the if statement is true, increament the counter by one.
    counter_datagap = counter_datagap + 1;

end % end of if statement that looks for the DATAGAPS.

% if the results files containing the analysis of the wide
beam only.
if strcmp(beams, beamstype1)

    % statistical analysis no:1.
    if strcmp(statanalysis, stattype1)

        % collect the total number for each event.
        total_aa_spikes(currentmonth) = counter_aa_spikes;
        total_aa_all(currentmonth) = counter_aa_all;
        total_pca(currentmonth) = counter_pca;
        total_noise(currentmonth) = counter_noise;
        total_datagap(currentmonth) = counter_datagap;
```

```
end % end of if statement for the statistical analysis no:1.

% statistical analysis no:2.
if strcmp(statanalysis, stattype2)

    % collect the total number of SPIKES - AAs.
    total_aa_spikes(currentmonth) = counter_aa_spikes;

end % end of if statement for the statistical analysis no:2.

% statistical analysis no:3.
if strcmp(statanalysis, stattype3)

    % collect the total number of ALL AAs.
    total_aa_all(currentmonth) = counter_aa_all;

end % end of if statement for the statistical analysis no:3.

% statistical analysis no:4.
if strcmp(statanalysis, stattype4)

    % collect the total number of PCAs.
    total_pca(currentmonth) = counter_pca;

end % end of if statement for the statistical analysis no:4.

% statistical analysis no:5.
if strcmp(statanalysis, stattype5)

    % collect the total number of NOISE.
    total_noise(currentmonth) = counter_noise;

end % end of if statement for the statistical analysis no:5.
```



```
% statistical analysis no:6.
if strcmp(statanalysis, statype6)

    % collect the total number of DATAGAPs.
    total_datagap(currentmonth) = counter_datagap;

end % end of if statement for the statistical analysis no:6.


end % end of if statement for the results file with the
widebeam.


% if the results files containing the analysis for all the
beams.
if strcmp(beams, beamstype2)

    % statistical analysis no:1.
    if strcmp(statanalysis, statype1)

        % collect the total number for each event.
        % apply the averaging factor.
        total_aa_spikes(currentmonth) = counter_aa_spikes / 50;
        total_aa_all(currentmonth) = counter_aa_all / 50;
        total_pca(currentmonth) = counter_pca / 50;
        total_noise(currentmonth) = counter_noise / 50;
        total_datagap(currentmonth) = counter_datagap / 50;

    end % end of if statement for the statistical analysis no:1.


    % statistical analysis no:2.
    if strcmp(statanalysis, statype2)
```

```
% collect the total number of SPIKES - AAs.
% apply the averaging factor.
total_aa_spikes(currentmonth) = counter_aa_spikes / 50;

end % end of if statement for the statistical analysis no:2.

% statistical analysis no:3.
if strcmp(statanalysis, stattype3)

    % collect the total number of ALL AAs.
    % apply the averaging factor.
    total_aa_all(currentmonth) = counter_aa_all / 50;

end % end of if statement for the statistical analysis no:3.

% statistical analysis no:4.
if strcmp(statanalysis, stattype4)

    % collect the total number of PCAs.
    % apply the averaging factor.
    total_pca(currentmonth) = counter_pca / 50;

end % end of if statement for the statistical analysis no:4.

% statistical analysis no:5.
if strcmp(statanalysis, stattype5)

    % collect the total number of NOISE.
    % apply the averaging factor.
    total_noise(currentmonth) = counter_noise / 50;

end % end of if statement for the statistical analysis no:5.
```

```
% statistical analysis no:6.
if strcmp(statanalysis, stattype6)

    % collect the total number of DATAGAPs.
    % apply the averaging factor.
    total_datagap(currentmonth) = counter_datagap / 50;

end % end of if statement for the statistical analysis no:6.

end % end of if statement for the results file with for all
the beams.

% increament the counter by one.
results_counter = results_counter + 1;

end % end of while loop for the results.

% reinitialise the counters that keep track of the number
% of different events that occur in one month.
counter_aa_spikes = 0;
counter_aa_all = 0;
counter_pca = 0;
counter_noise = 0;
counter_datagap = 0;

end % end of for loop for the months.

% each variable is an array 1x12 and in each column keeps the
% number of occrrences of a single event.
total_aa_spikes;
total_aa_all;
total_pca;
total_noise;
```

```
total_datagap;

% these information will be needed later for the histograms.
totalmonths = [1 2 3 4 5 6 7 8 9 10 11 12];

% plot a histogram for all the events.
if strcmp(statanalysis, stattype1)

% open a new figure for the histogram.
h_all = figure;

% put the total number of Spiky AAs, the total number of All
AAs,
% the total number of PCAs, the total number of Noise and the
% total number of Datagaps into a single array.
total_all = ...
    [total_aa_spikes; total_aa_all; ...
    total_pca; total_noise; total_datagap];

% plot the number of all the events in a histogram.
bar(datenum(1, totalmonths, 1), total_all', 0.9, 'grouped');

% add useful information on the histogram.
titlestr = ...
    char(['All Events Histogram.' ...
        ' Year ' strftime(results_starttime, '%Y')]);
title(titlestr);
datetick('x', 'mmm');
ylabel('Number of Events');
legend('Spikes', 'All AAs', 'PCAs', 'Noise', 'Datagaps');
grid
set(h_all, 'name', 'Statistical Analysis III');

% only for the case of all the events, plot 12 histograms.
% each histogram will contain the number of all the events
% for every month.
```

```

% open a new figure for the sub-histograms.
h_sub = figure;

% sub-plots for all the events for the monts 01-12.
for submonth = totalmonths

    % sub-plot for all the events for the month 01.
    subplot(3, 4, submonth);
    bar(total_all(:, submonth));

    % add useful information on the sub-histograms.
    subdate = time([year submonth 1 0 0 0]);
    titlestr = ...
        char(['Month ' strftime(subdate , '%m') ...
            ' Year ' strftime(results_starttime, '%Y')]);
    title(titlestr);
    ylabel('Number of Events');
    grid
    set(h_sub, 'name', 'Statistical Analysis III');

end % end of for loop for the sub-histograms.

% add extra information on the whole figure of sub-histograms.
legend('1-Spikes', '2-All AAs', '3-PCAs', '4-Noise', '5-
Datagaps');

% print useful information on the Matlab Workspace.
fprintf('\nStatistical Analysis III, Year %d\n', year);

for i = loadmonths

    fprintf('\nMonth %d\n', i);
    fprintf('Total Number of Spikes      : %.1f\n',
total_aa_spikes(:, i));
    fprintf('Total Number of All AAs      : %.1f\n', total_aa_all(:,

```

```
i));  
fprintf('Total Number of PCAs      : %.1f\n', total_pca(:,  
i));  
fprintf('Total Number of Noise      : %.1f\n', total_noise(:,  
i));  
fprintf('Total Number of Datagaps    : %.1f\n', total_datagap(:,  
i));  
  
end % end of for loop for the information on the workspace.  
  
end % end of the histogram for all the events.  
  
% plot a histogram for the Spiky AAs.  
if strcmp(statanalysis, stattype2)  
  
% put the total number of the Spiky AAs into a single array.  
total_all = [total_aa_spikes];  
  
% open a new figure for the histogram.  
h_aa_spikes = figure;  
  
% plot the number of the Spiky AAs in a histogram.  
bar(datenum(1, totalmonths, 1), total_all');  
  
% add useful information on the histogram.  
titlestr = ...  
    char(['Spikes Histogram.' ...  
        ' Year ' strftime(results_starttime, '%Y')]);  
title(titlestr);  
datetick('x', 'mmm');  
ylabel('Number of Events');  
legend('Spikes');  
grid  
set(h_aa_spikes, 'name', 'Statistical Analysis III');  
  
% print useful information on the Matlab Workspace.
```

```
fprintf('\nStatistical Analysis III, Year %d\n', year);

for i = loadmonths

    fprintf('\nMonth %d\n', i);
    fprintf('Total Number of Spikes : %.1f\n', total_aa_spikes(:,
i));

end % end of for loop for the information on the workspace.

end % end of the histogram for the Spikes.

% plot a histogram for All the AAs.
if strcmp(statanalysis, stattype3)

% put the total number of All the AAs into a single array.
total_all = [total_aa_all];

% open a new figure for the histogram.
h_aa_all = figure;

% plot the number of All the AAs in a histogram.
bar(datenum(1, totalmonths, 1), total_all');

% add useful information on the histogram.
titlestr = ...
    char(['All Auroral Absorptions Histogram.' ...
        ' Year ' strftime(results_starttime, '%Y')]);
title(titlestr);
datetick('x', 'mmm');
ylabel('Number of Events');
legend('All AAs');
grid
set(h_aa_all, 'name', 'Statistical Analysis III');

% print useful information on the Matlab Workspace.
```

```
fprintf('\nStatistical Analysis III, Year %d\n', year);

for i = loadmonths

    fprintf('\nMonth %d\n', i);
    fprintf('Total Number of All AAs : %.1f\n', total_aa_all(:,
i));

end % end of for loop for the information on the workspace.

end % end of the histogram for all the AAs.

% plot a histogram for the PCAs.
if strcmp(statanalysis, stattype4)

% put the total number of the PCAs into a single array.
total_all = [total_pca];

% open a new figure for the histogram.
h_pca = figure;

% plot the number of the PCAs in a histogram.
bar(datenum(1, totalmonths, 1), total_all');

% add useful information on the histogram.
titlestr = ...
    char(['Polar Cap Absorptions Histogram.' ...
        ' Year ' strftime(results_starttime, '%Y')]);
title(titlestr);
datetick('x', 'mmm');
ylabel('Number of Events');
legend('PCAs');
grid
set(h_pca, 'name', 'Statistical Analysis III');

% print useful information on the Matlab Workspace.
```



```
fprintf('\nStatistical Analysis III, Year %d\n', year);

for i = loadmonths

    fprintf('\nMonth %d\n', i);
    fprintf('Total Number of PCAs : %.1f\n', total_pca(:, i));

end % end of for loop for the information on the workspace.

end % end of the histogram for the PCAs.


% plot a histogram for the Noise.
if strcmp(statanalysis, stattype5)

    % put the total number of the Noise into a single array.
    total_all = [total_noise];

    % open a new figure for the histogram.
    h_noise = figure;

    % plot the number of the Noise in a histogram.
    bar(datenum(1, totalmonths, 1), total_all');

    % add useful information on the histogram.
    titlestr = ...
        char(['Noise Histogram.' ...
            ' Year ' strftime(results_starttime, '%Y')]);
    title(titlestr);
    datetick('x', 'mmm');
    ylabel('Number of Events');
    legend('Noise');
    grid
    set(h_noise, 'name', 'Statistical Analysis III');

    % print useful information on the Matlab Workspace.
    fprintf('\nStatistical Analysis III, Year %d\n', year);
```

```
for i = loadmonths

fprintf('\nMonth %d\n', i);
fprintf('Total Number of Noise : %.1f\n', total_noise(:, i));

end % end of for loop for the information on the workspace.

end % end of the histogram for the Noise.


% plot a histogram for the Datagaps.
if strcmp(statanalysis, stattype6)

% put the total number of Datagaps into a single array.
total_all = [total_datagap];

% open a new figure for the histogram.
h_datagap = figure;

% plot the number of the Datagaps in a histogram.
bar(datenum(1, totalmonths, 1), total_all');

% add useful information on the histogram.
titlestr = ...
    char(['Datagaps Histogram.' ...
        ' Year ' strftime(results_starttime, '%Y')]);
title(titlestr);
datetick('x', 'mmm');
ylabel('Number of Events');
legend('Datagaps');
grid
set(h_datagap, 'name', 'Statistical Analysis III');

% print useful information on the Matlab Workspace.
fprintf('\nStatistical Analysis III, Year %d\n', year);
```

```
for i = loadmonths

fprintf('\nMonth %d\n', i);
fprintf('Total Number of Datagaps : %.1f\n', total_datagap(:,
i));

end % end of for loop for the information on the workspace.

end % end of the histogram for the Datagaps.


% if error occur catch the next commands.
catch

    % print a message on the Matlab Workspace.
    fprintf('\nThere are No results available for the Year
%d\n', year);

end % end of try-catch statement.
```

APPENDIX B

LONG TERM STATISTICAL STUDIES OF IONOSPHERIC ABSORPTION

STATISTICAL ANALYSIS RESULTS - TABLES

Table 5.1.1: Statistical Analysis I, Summary Results Table Year 1994

Statistical Analysis I						
Year	Number of Auroral Absorptions / Number of Spikes					
1994	Depending on the Time of their Occurrence.					
Month	00:00:00	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00
	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00	24:00:00
01	-	-	-	-	-	-
02	-	-	-	-	-	-
03	-	-	-	-	-	-
04	-	-	-	-	-	-
05	-	-	-	-	-	-
06	-	-	-	-	-	-
07	-	-	-	-	-	-
08	-	-	-	-	-	-
09	-	-	-	-	-	-
10	35 / 01	30 / 02	39 / 05	20 / 04	28 / 08	48 / 04
11	24 / 01	28 / 02	30 / 04	12 / 00	19 / 04	36 / 12
12	32 / 01	30 / 06	34 / 00	18 / 04	21 / 07	41 / 06

Table 5.1.2: Statistical Analysis I, Summary Results Table Year 1995

Statistical Analysis I						
Year	Number of Auroral Absorptions / Number of Spikes					
1995	Depending on the Time of their Occurrence.					
Month	00:00:00	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00
	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00	24:00:00
01	27 / 05	30 / 03	28 / 08	12 / 04	10 / 01	37 / 02
02	46 / 07	41 / 05	33 / 04	12 / 03	21 / 06	35 / 03
03	42 / 05	46 / 08	41 / 02	16 / 05	23 / 04	32 / 01
04	18 / 03	14 / 01	30 / 03	11 / 02	26 / 00	25 / 03
05	39 / 05	27 / 02	54 / 08	12 / 05	30 / 04	45 / 03
06	22 / 02	14 / 00	18 / 04	10 / 01	23 / 01	08 / 00
07	14 / 01	09 / 01	07 / 00	19 / 05	02 / 02	02 / 00
08	10 / 00	04 / 00	06 / 01	01 / 01	01 / 00	04 / 00
09	11 / 01	22 / 02	27 / 04	03 / 00	14 / 03	18 / 03
10	42 / 05	34 / 04	34 / 01	16 / 02	28 / 03	59 / 04
11	16 / 02	17 / 01	18 / 03	05 / 01	10 / 02	20 / 03
12	08 / 01	22 / 06	07 / 00	04 / 01	07 / 03	23 / 09

Table 5.1.3: Statistical Analysis I, Summary Results Table Year 1996

Statistical Analysis I						
Year 1996	Number of Auroral Absorptions / Number of Spikes Depending on the Time of their Occurrence.					
Month	00:00:00	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00
	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00	24:00:00
01	25 / 04	04 / 00	13 / 06	08 / 03	04 / 02	32 / 04
02	31 / 02	14 / 02	19 / 02	08 / 01	06 / 01	32 / 01
03	37 / 03	27 / 02	39 / 05	11 / 03	14 / 01	41 / 07
04	37 / 02	22 / 07	24 / 02	15 / 04	21 / 02	34 / 04
05	14 / 03	17 / 05	12 / 03	02 / 01	15 / 03	29 / 07
06	04 / 00	10 / 00	00 / 00	00 / 00	00 / 00	01 / 00
07	18 / 01	18 / 02	12 / 01	03 / 00	02 / 00	18 / 03
08	55 / 12	47 / 07	45 / 03	35 / 05	48 / 08	48 / 10
09	34 / 04	52 / 10	44 / 05	26 / 03	33 / 06	47 / 06
10	30 / 04	23 / 00	29 / 02	11 / 00	13 / 01	35 / 09
11	19 / 00	11 / 02	24 / 03	07 / 02	11 / 02	14 / 01
12	09 / 00	21 / 02	20 / 01	05 / 02	13 / 00	35 / 04

Table 5.1.4: Statistical Analysis I, Summary Results Table Year 1997

Statistical Analysis I						
Year	Number of Auroral Absorptions / Number of Spikes					
1997	Depending on the Time of their Occurrence.					
Month	00:00:00	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00
	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00	24:00:00
01	18 / 04	15 / 01	07 / 00	05 / 00	06 / 05	24 / 03
02	20 / 04	17 / 00	14 / 02	13 / 03	15 / 04	21 / 01
03	15 / 01	23 / 02	13 / 01	04 / 01	15 / 03	22 / 01
04	26 / 05	27 / 04	08 / 02	10 / 01	01 / 00	11 / 00
05	17 / 02	16 / 03	10 / 00	13 / 00	29 / 04	36 / 02
06	23 / 01	09 / 00	05 / 02	05 / 00	11 / 01	26 / 04
07	08 / 01	21 / 06	14 / 00	26 / 02	11 / 03	18 / 01
08	23 / 04	11 / 01	11 / 01	07 / 00	09 / 02	20 / 04
09	41 / 02	35 / 03	33 / 05	29 / 03	14 / 03	29 / 07
10	17 / 03	16 / 02	16 / 01	09 / 01	18 / 05	23 / 04
11	20 / 00	28 / 04	10 / 02	04 / 02	07 / 03	10 / 01
12	03 / 01	03 / 00	05 / 00	05 / 00	00 / 00	02 / 00

Table 5.1.5: Statistical Analysis I, Summary Results Table Year 1998

Statistical Analysis I						
Year	Number of Auroral Absorptions / Number of Spikes					
1998	Depending on the Time of their Occurrence.					
Month	00:00:00	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00
	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00	24:00:00
01	14 / 03	07 / 00	14 / 00	06 / 01	07 / 02	06 / 00
02	33 / 05	14 / 02	21 / 01	04 / 00	08 / 04	23 / 02
03	55 / 07	36 / 04	35 / 05	05 / 00	08 / 01	32 / 03
04	31 / 04	28 / 01	57 / 12	52 / 05	08 / 00	24 / 04
05	48 / 07	43 / 08	73 / 10	41 / 05	33 / 08	59 / 02
06	40 / 03	31 / 04	33 / 05	11 / 03	22 / 04	46 / 08
07	12 / 01	12 / 03	25 / 05	23 / 05	26 / 03	24 / 06
08	42 / 11	24 / 04	68 / 16	66 / 09	39 / 02	46 / 07
09	19 / 01	34 / 04	30 / 04	10 / 01	10 / 01	20 / 02
10	49 / 07	42 / 07	24 / 05	18 / 10	27 / 05	28 / 01
11	28 / 03	30 / 03	28 / 05	16 / 03	10 / 03	31 / 06
12	15 / 04	13 / 02	13 / 02	00 / 00	05 / 01	05 / 00

Table 5.1.6: Statistical Analysis I, Summary Results Table Year 1999

Statistical Analysis I						
Year 1999	Number of Auroral Absorptions / Number of Spikes Depending on the Time of their Occurrence.					
Month	00:00:00	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00
	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00	24:00:00
01	16 / 01	16 / 00	29 / 05	19 / 04	09 / 04	20 / 01
02	08 / 01	14 / 02	36 / 00	03 / 00	09 / 00	08 / 02
03	49 / 08	39 / 07	46 / 10	10 / 01	09 / 01	35 / 02
04	37 / 03	40 / 06	53 / 11	20 / 03	22 / 04	29 / 01
05	31 / 04	30 / 05	41 / 10	48 / 07	17 / 03	23 / 02
06	53 / 13	67 / 13	66 / 08	69 / 11	60 / 14	60 / 06
07	27 / 10	36 / 12	31 / 10	16 / 05	30 / 16	40 / 11
08	55 / 08	40 / 05	42 / 06	07 / 01	21 / 02	40 / 08
09	41 / 04	56 / 05	41 / 08	11 / 01	14 / 02	45 / 06
10	47 / 02	53 / 05	68 / 19	08 / 01	23 / 04	46 / 08
11	32 / 04	58 / 06	319 / 177	79 / 40	20 / 01	40 / 07
12	31 / 02	61 / 11	57 / 07	17 / 02	20 / 02	30 / 02

Table 5.1.7: Statistical Analysis I, Summary Results Table Year 2000

Statistical Analysis I						
Year	Number of Auroral Absorptions / Number of Spikes					
2000	Depending on the Time of their Occurrence.					
Month	00:00:00	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00
	04:00:00	08:00:00	12:00:00	16:00:00	20:00:00	24:00:00
01	20 / 03	46 / 08	47 / 07	10 / 01	16 / 02	31 / 03
02	19 / 02	48 / 05	66 / 15	25 / 06	13 / 03	35 / 04
03	21 / 02	68 / 13	149 / 51	41 / 05	07 / 03	14 / 03
04	54 / 08	68 / 08	96 / 19	31 / 05	16 / 01	41 / 03
05	41 / 04	48 / 04	77 / 16	53 / 07	32 / 02	48 / 07
06	56 / 09	46 / 09	38 / 05	21 / 02	36 / 04	24 / 02
07	-	-	-	-	-	-
08	-	-	-	-	-	-
09	-	-	-	-	-	-
10	-	-	-	-	-	-
11	-	-	-	-	-	-
12	-	-	-	-	-	-

Table 5.2.1: Statistical Analysis II, Summary Results Table Year 1994

Statistical Analysis II						
Year	Number of Auroral Absorptions					
1994	Depending on their Duration.					
Month	>2mins <6mins	>6mins <30mins	>30mins <1hours	>1hours <2hours	>2hours <3hours	>3hours <5hours
01	-	-	-	-	-	-
02	-	-	-	-	-	-
03	-	-	-	-	-	-
04	-	-	-	-	-	-
05	-	-	-	-	-	-
06	-	-	-	-	-	-
07	-	-	-	-	-	-
08	-	-	-	-	-	-
09	-	-	-	-	-	-
10	24	111	37	18	05	00
11	23	87	15	13	07	00
12	24	97	25	20	05	00

Table 5.2.2: Statistical Analysis II, Summary Results Table Year 1995

Statistical Analysis II						
Year	Number of Auroral Absorptions					
1995	Depending on their Duration.					
Month	>2mins <6mins	>6mins <30mins	>30mins <1hours	>1hours <2hours	>2hours <3hours	>3hours <5hours
01	23	80	27	10	04	00
02	28	115	26	14	02	00
03	25	119	28	17	07	00
04	12	79	15	15	01	00
05	27	111	38	23	07	00
06	08	54	15	09	08	00
07	09	30	07	06	01	00
08	02	17	05	02	00	00
09	13	52	10	11	06	00
10	19	135	27	26	03	00
11	12	51	08	12	03	00
12	20	40	05	04	01	00

Table 5.2.3: Statistical Analysis II, Summary Results Table Year 1996

Statistical Analysis II						
Year	Number of Auroral Absorptions					
1996	Depending on their Duration.					
Month	>2mins <6mins	>6mins <30mins	>30mins <1hours	>1hours <2hours	>2hours <3hours	>3hours <5hours
01	19	48	10	07	01	00
02	09	65	15	17	01	00
03	21	91	32	16	05	00
04	21	77	25	23	05	00
05	22	48	12	07	00	00
06	00	14	00	01	00	00
07	07	48	11	05	00	00
08	45	176	31	15	08	00
09	34	143	26	21	12	00
10	16	80	16	22	04	00
11	10	50	16	09	01	00
12	09	76	07	08	03	00

Table 5.2.4: Statistical Analysis II, Summary Results Table Year 1997

Statistical Analysis II						
Year	Number of Auroral Absorptions					
1997	Depending on their Duration.					
Month	>2mins <6mins	>6mins <30mins	>30mins <1hours	>1hours <2hours	>2hours <3hours	>3hours <5hours
01	13	42	09	08	02	00
02	14	58	15	10	01	00
03	09	55	09	13	05	00
04	12	50	11	07	03	00
05	11	86	18	04	02	00
06	08	52	11	06	02	00
07	13	66	09	09	00	00
08	12	48	09	09	02	00
09	23	115	23	13	04	00
10	16	59	10	11	03	00
11	12	52	06	06	03	00
12	01	14	00	02	01	00

Table 5.2.5: Statistical Analysis II, Summary Results Table Year 1998

Statistical Analysis II						
Year	Number of Auroral Absorptions					
1998	Depending on their Duration.					
Month	>2mins <6mins	>6mins <30mins	>30mins <1hours	>1hours <2hours	>2hours <3hours	>3hours <5hours
01	06	34	05	06	01	00
02	14	69	17	02	01	00
03	20	102	25	18	02	00
04	26	122	36	10	05	00
05	40	174	47	26	07	00
06	27	121	24	10	01	00
07	23	72	18	04	04	00
08	49	187	27	14	04	00
09	13	80	17	09	04	00
10	35	119	17	12	04	00
11	23	86	20	09	01	00
12	09	30	05	05	02	00

Table 5.2.6: Statistical Analysis II, Summary Results Table Year 1999

Statistical Analysis II						
Year	Number of Auroral Absorptions					
1999	Depending on their Duration.					
Month	>2mins <6mins	>6mins <30mins	>30mins <1hours	>1hours <2hours	>2hours <3hours	>3hours <5hours
01	15	61	21	08	03	00
02	05	44	14	12	02	00
03	29	93	33	23	10	00
04	28	112	28	22	07	00
05	31	123	19	12	05	00
06	65	272	19	15	03	00
07	64	99	12	04	01	00
08	30	132	24	15	02	00
09	26	124	24	23	06	00
10	39	128	42	28	05	00
11	235	246	36	22	07	00
12	26	144	21	17	03	00

Table 5.2.7: Statistical Analysis II, Summary Results Table Year 2000

Statistical Analysis II						
Year 2000	Number of Auroral Absorptions Depending on their Duration.					
Month	>2mins <6mins	>6mins <30mins	>30mins <1hours	>1hours <2hours	>2hours <3hours	>3hours <5hours
01	24	104	21	15	04	00
02	35	115	22	20	13	00
03	77	178	31	11	02	00
04	44	200	37	18	06	00
05	40	182	39	22	11	00
06	31	150	22	14	02	00
07	-	-	-	-	-	-
08	-	-	-	-	-	-
09	-	-	-	-	-	-
10	-	-	-	-	-	-
11	-	-	-	-	-	-
12	-	-	-	-	-	-

Table 5.3.1: Statistical Analysis III, Summary Results Table Year 1994

Statistical Analysis III					
Year 1994	Number of Events				
Month	Spikes	AA	PCA	Noise	DataGaps
01 - January	-	-	-	-	-
02 - February	-	-	-	-	-
03 - March	-	-	-	-	-
04 - April	-	-	-	-	-
05 - May	-	-	-	-	-
06 - June	-	-	-	-	-
07 - July	-	-	-	-	-
08 - August	-	-	-	-	-
09 - September	-	-	-	-	-
10 - October	24	200	01	12	12
11 - November	23	149	00	14	13
12 - December	24	176	00	01	05

Table 5.3.2: Statistical Analysis III, Summary Results Table Year 1995

Statistical Analysis III					
Year 1995	Number of Events				
Month	Spikes	AA	PCA	Noise	DataGaps
01 - January	23	144	00	01	17
02 - February	28	188	00	02	08
03 - March	25	200	00	45	04
04 - April	12	124	02	10	08
05 - May	27	207	01	05	51
06 - June	08	95	01	09	28
07 - July	09	53	01	37	05
08 - August	02	26	00	00	05
09 - September	13	95	00	06	15
10 - October	19	213	00	06	23
11 - November	12	86	00	01	16
12 - December	20	71	01	01	05

Table 5.3.3: Statistical Analysis III, Summary Results Table Year 1996

Statistical Analysis III					
Year 1996	Number of Events				
Month	Spikes	AA	PCA	Noise	DataGaps
01 - January	19	86	00	03	09
02 - February	09	110	00	02	20
03 - March	21	169	02	00	26
04 - April	21	153	00	00	22
05 - May	22	89	00	25	35
06 - June	00	15	00	14	20
07 - July	07	71	00	26	21
08 - August	45	278	00	59	10
09 - September	34	236	02	01	28
10 - October	16	141	01	01	26
11 - November	10	86	00	12	23
12 - December	09	103	00	01	11

Table 5.3.4: Statistical Analysis III, Summary Results Table Year 1997

Statistical Analysis III					
Year 1997	Number of Events				
Month	Spikes	AA	PCA	Noise	DataGaps
01 - January	13	75	00	01	17
02 - February	14	100	00	01	30
03 - March	09	92	00	04	19
04 - April	12	83	00	12	29
05 - May	11	121	00	19	11
06 - June	08	79	00	59	16
07 - July	13	98	01	33	30
08 - August	12	81	00	17	24
09 - September	23	181	00	17	24
10 - October	16	99	01	01	12
11 - November	12	79	01	04	09
12 - December	01	18	00	00	25

Table 5.3.5: Statistical Analysis III, Summary Results Table Year 1998

Statistical Analysis III					
Year 1998	Number of Events				
Month	Spikes	AA	PCA	Noise	DataGaps
01 - January	06	54	00	00	54
02 - February	14	103	00	00	119
03 - March	20	171	00	02	109
04 - April	26	200	04	47	112
05 - May	40	297	00	203	115
06 - June	27	183	00	113	123
07 - July	23	122	01	80	39
08 - August	49	285	07	109	67
09 - September	13	123	00	111	22
10 - October	35	188	01	06	132
11 - November	23	143	00	24	35
12 - December	09	51	00	44	42

Table 5.3.6: Statistical Analysis III, Summary Results Table Year 1999

Statistical Analysis III					
Year 1999	Number of Events				
Month	Spikes	AA	PCA	Noise	DataGaps
01 - January	15	109	00	16	17
02 - February	05	78	00	54	09
03 - March	29	188	00	26	16
04 - April	28	201	00	11	17
05 - May	31	190	00	165	14
06 - June	65	375	01	191	18
07 - July	64	180	00	242	189
08 - August	30	205	00	73	76
09 - September	26	208	01	59	23
10 - October	39	245	01	28	23
11 - November	235	548	00	278	17
12 - December	26	216	00	79	22

Table 5.3.7: Statistical Analysis III, Summary Results Table Year 2000

Statistical Analysis III					
Year 2000	Number of Events				
Month	Spikes	AA	PCA	Noise	DataGaps
01 - January	24	170	01	14	15
02 - February	35	206	00	79	16
03 - March	77	300	00	367	18
04 - April	44	306	01	120	14
05 - May	40	299	02	157	11
06 - June	31	221	02	268	37
07 - July	-	-	-	-	-
08 - August	-	-	-	-	-
09 - September	-	-	-	-	-
10 - October	-	-	-	-	-
11 - November	-	-	-	-	-
12 - December	-	-	-	-	-

APPENDIX C

LONG TERM STATISTICAL STUDIES OF IONOSPHERIC ABSORPTION

STATISTICAL ANALYSIS RESULTS - HISTOGRAMS

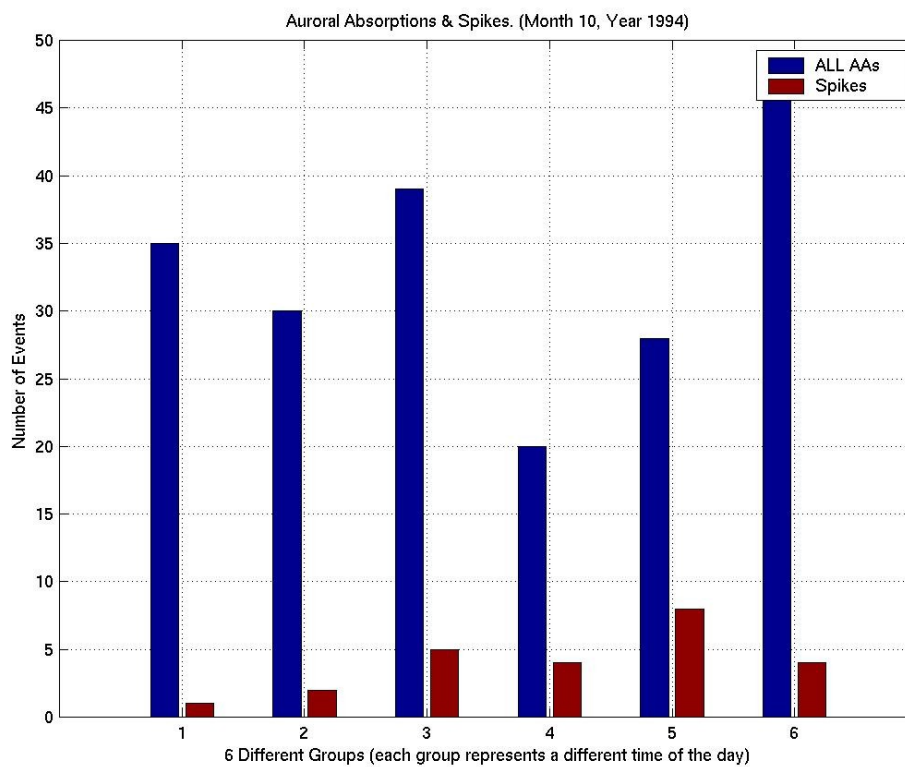


Figure 5.1.1: Statistical Analysis I, Month 10 Year 1994.

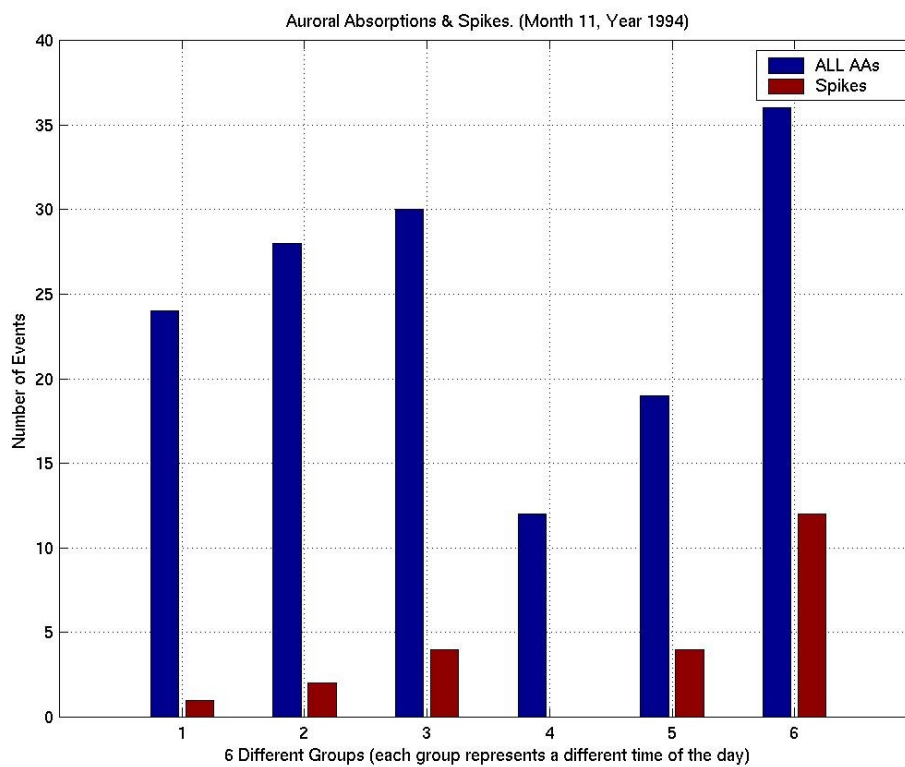


Figure 5.1.2: Statistical Analysis I, Month 11 Year 1994.

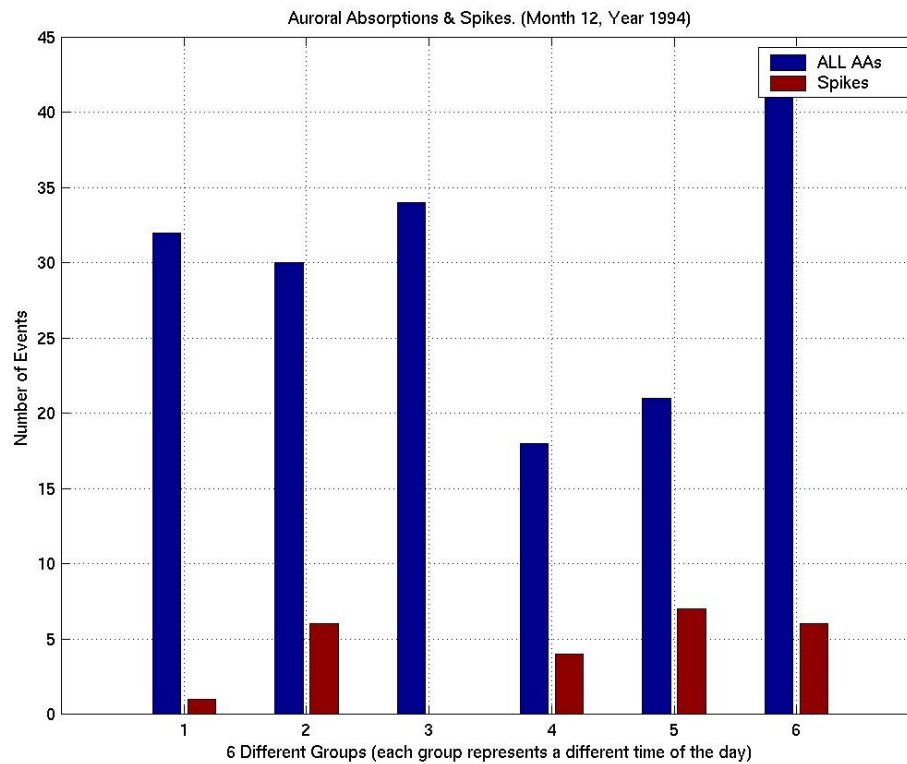


Figure 5.1.3: Statistical Analysis I, Month 12 Year 1994.

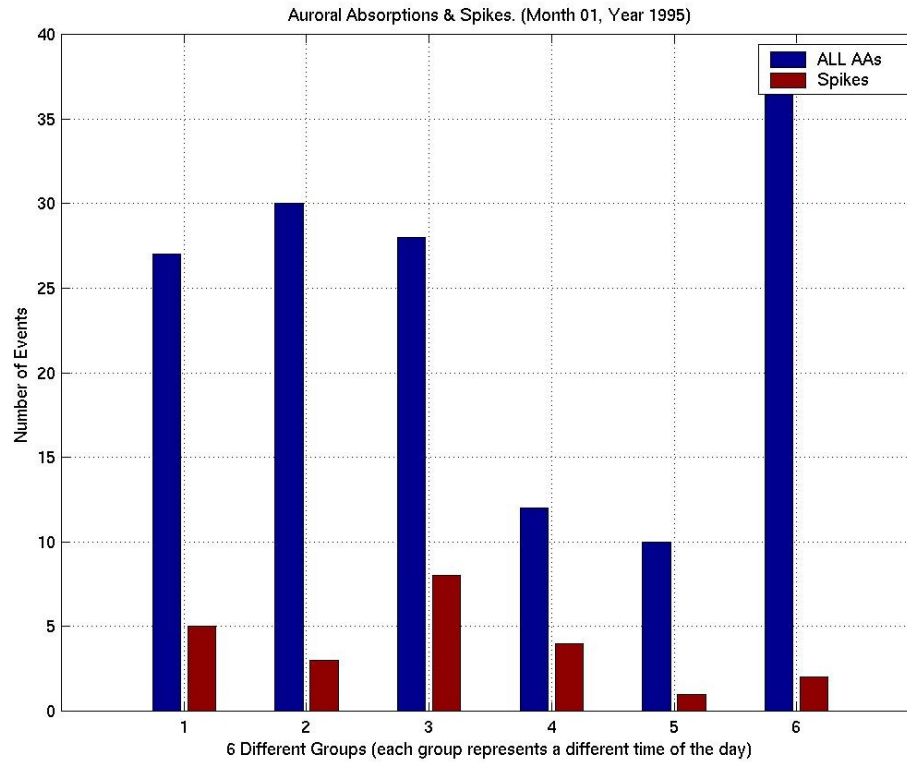


Figure 5.1.4: Statistical Analysis I, Month 01 Year 1995.

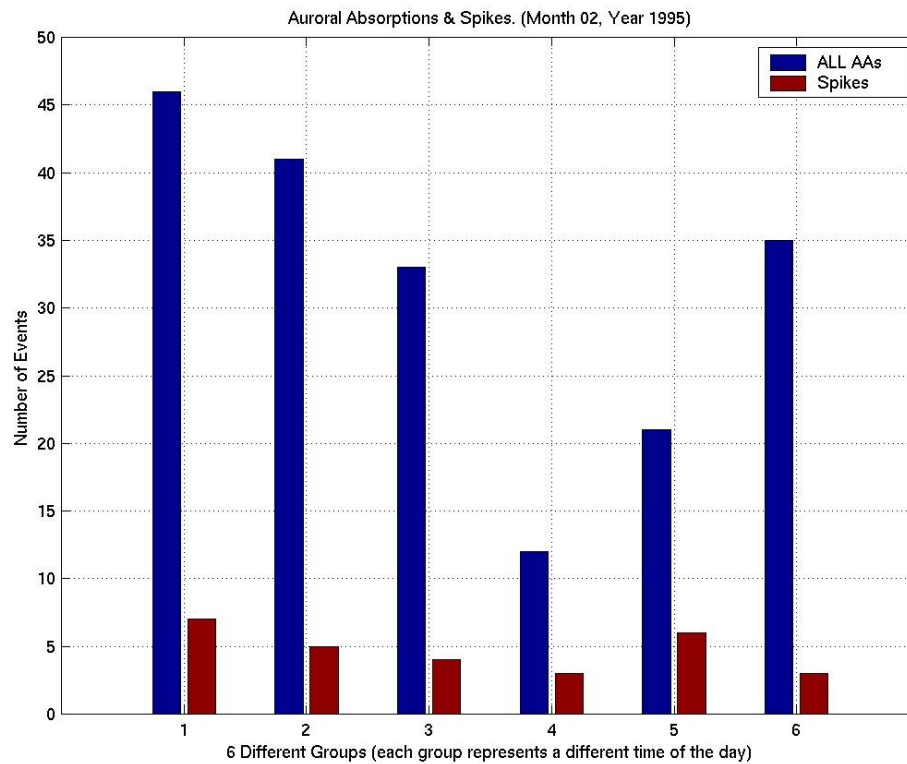


Figure 5.1.5: Statistical Analysis I, Month 02 Year 1995.

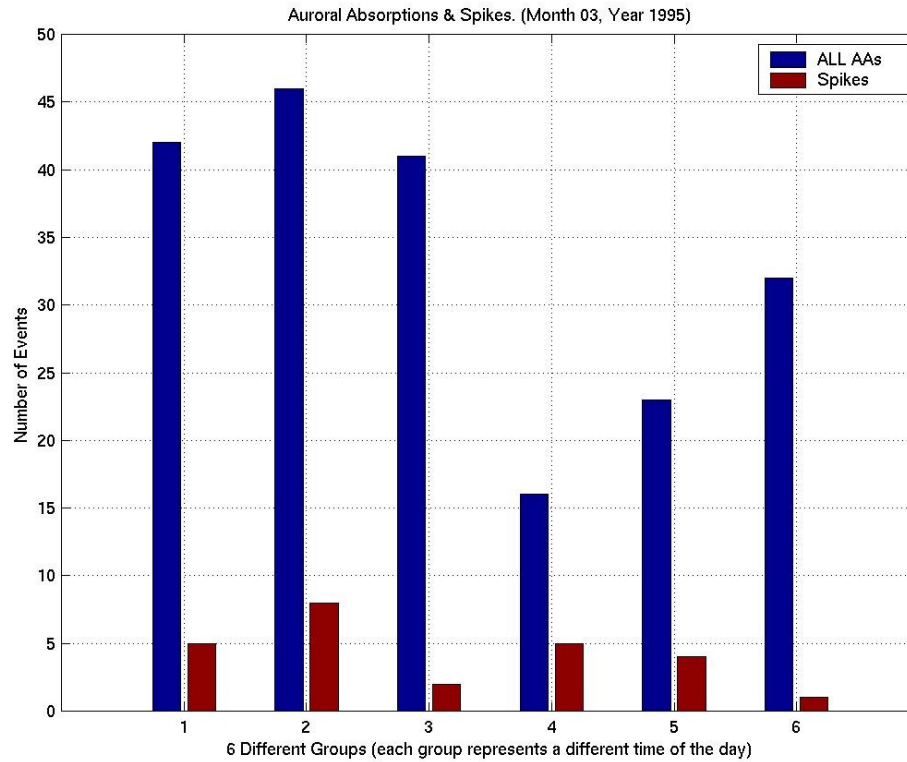


Figure 5.1.6: Statistical Analysis I, Month 03 Year 1995.

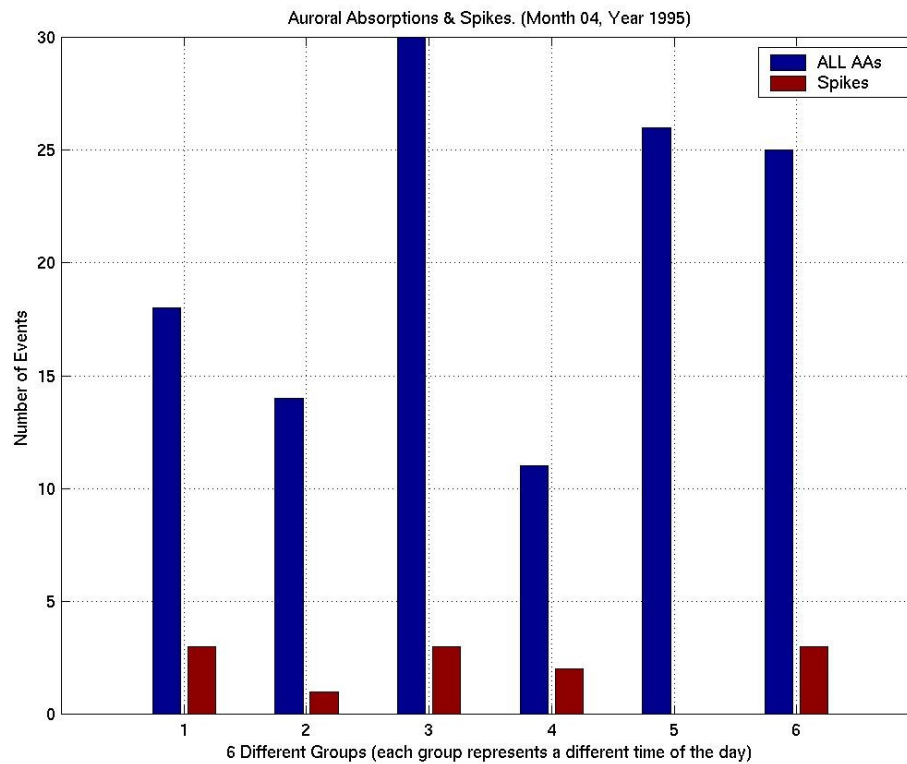


Figure 5.1.7: Statistical Analysis I, Month 04 Year 1995.

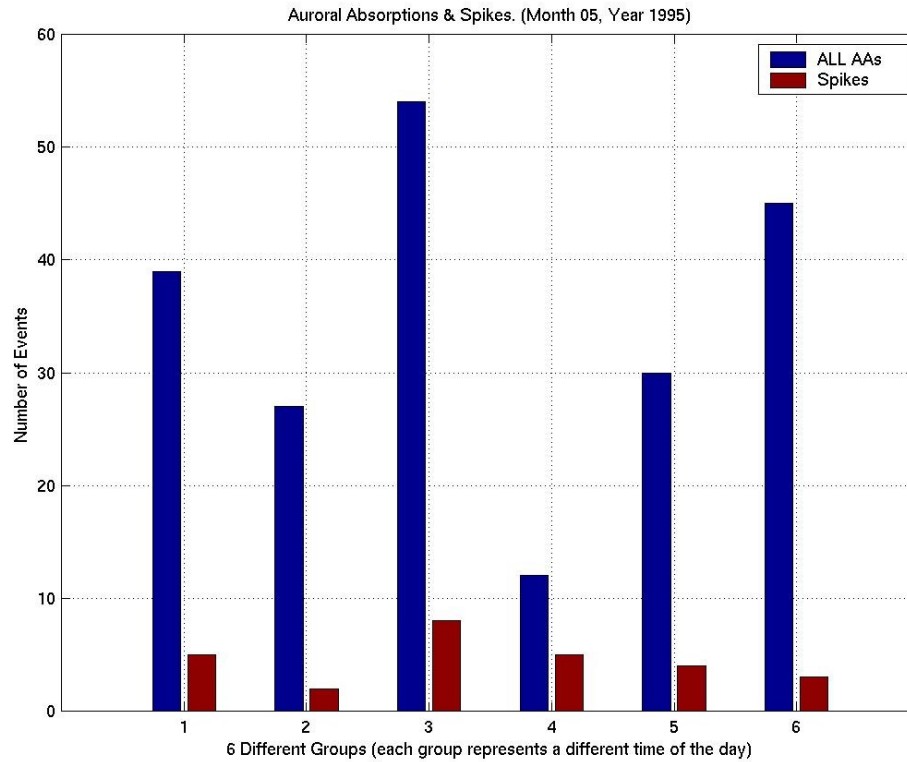


Figure 5.1.8: Statistical Analysis I, Month 05 Year 1995.

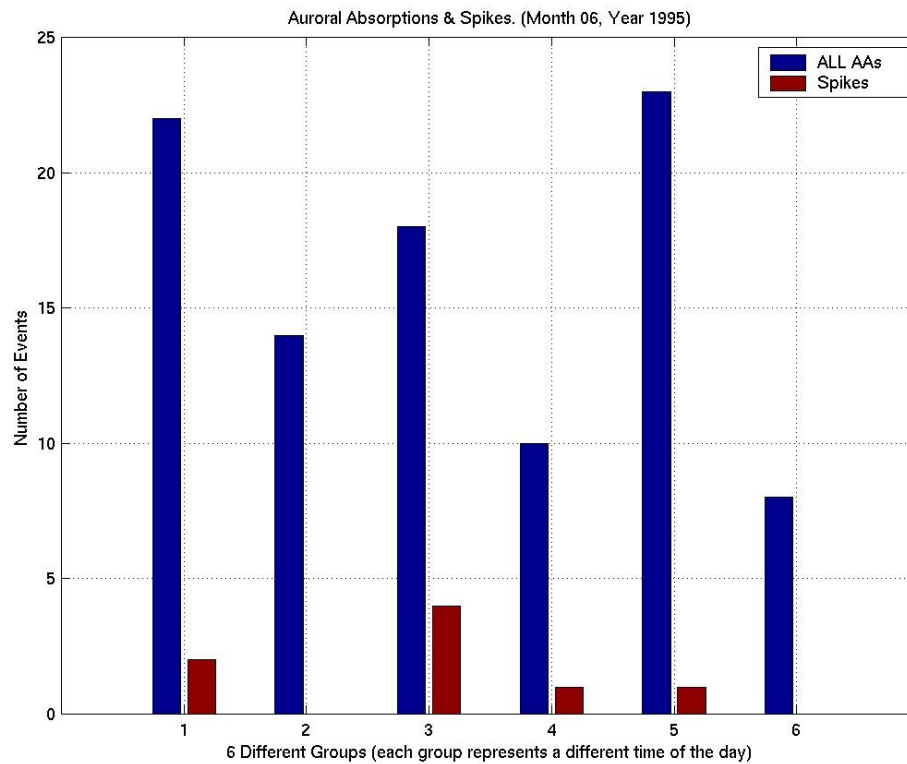


Figure 5.1.9: Statistical Analysis I, Month 06 Year 1995.

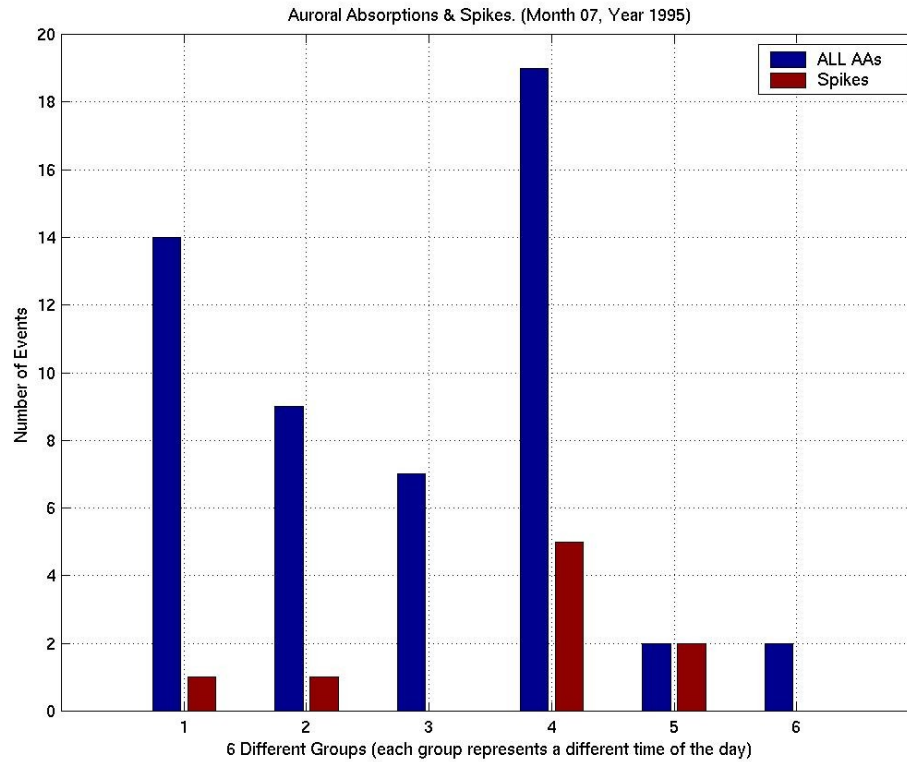


Figure 5.1.10: Statistical Analysis I, Month 07 Year 1995.

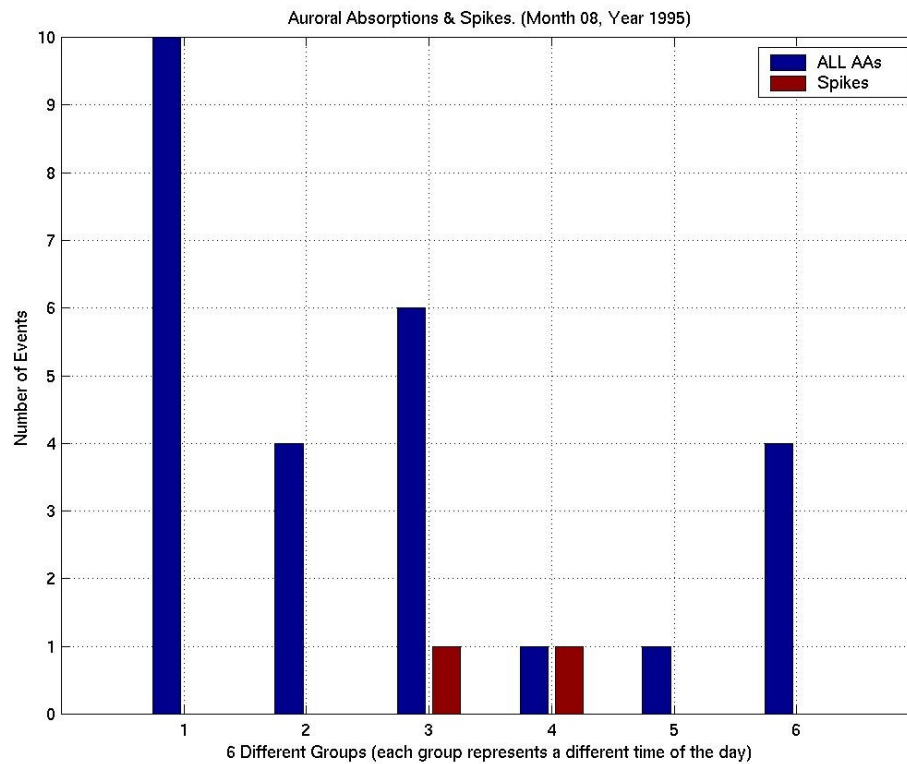


Figure 5.1.11: Statistical Analysis I, Month 08 Year 1995.

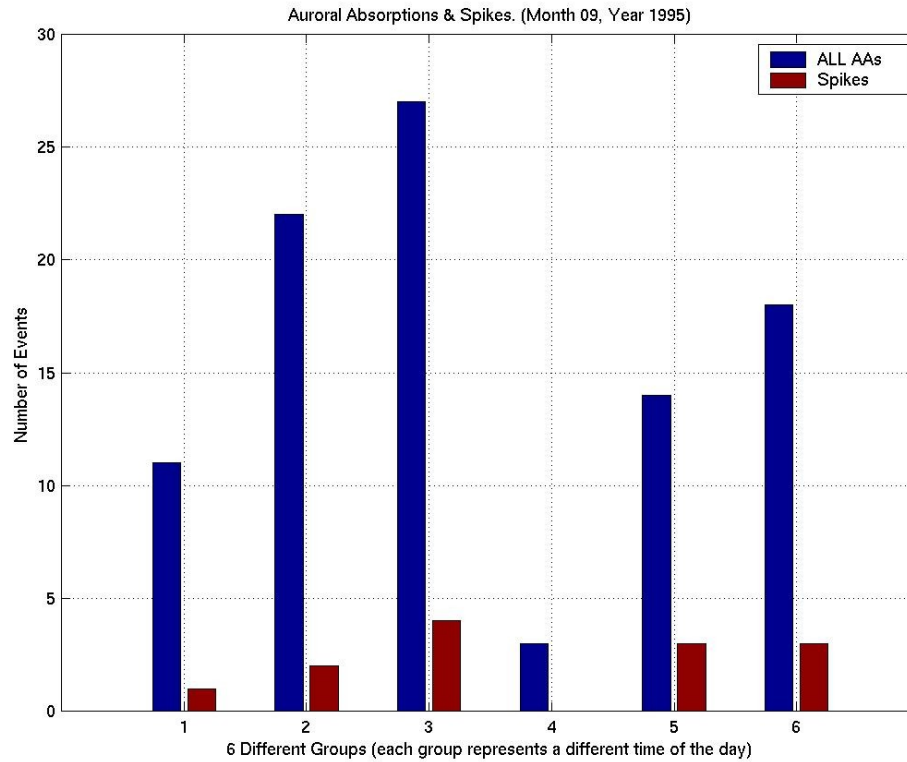


Figure 5.1.12: Statistical Analysis I, Month 09 Year 1995.

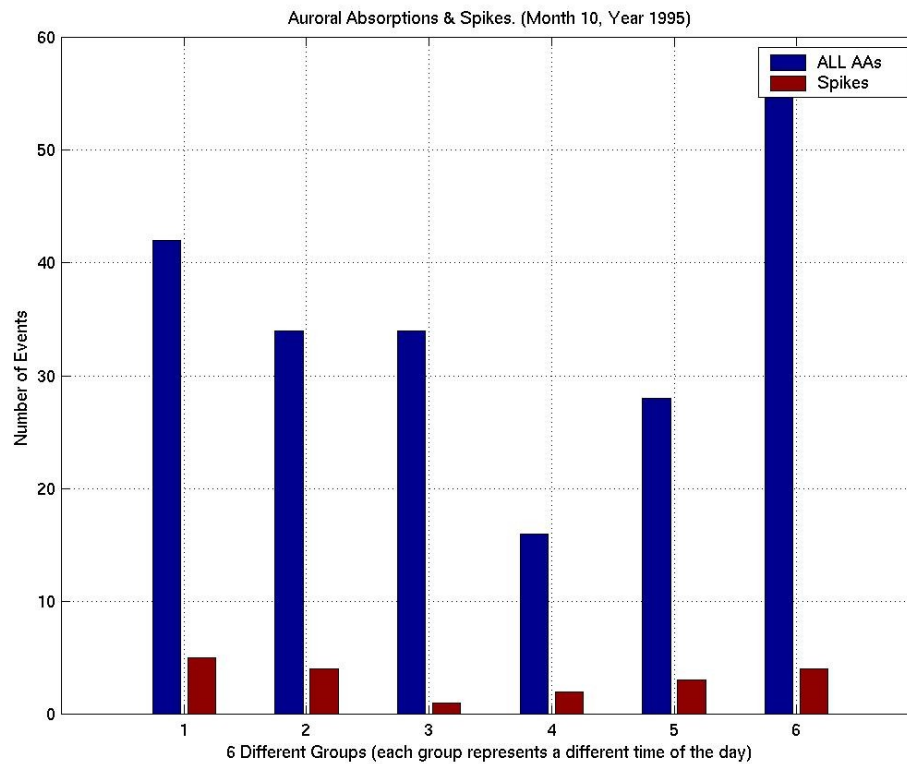


Figure 5.1.13: Statistical Analysis I, Month 10 Year 1995.

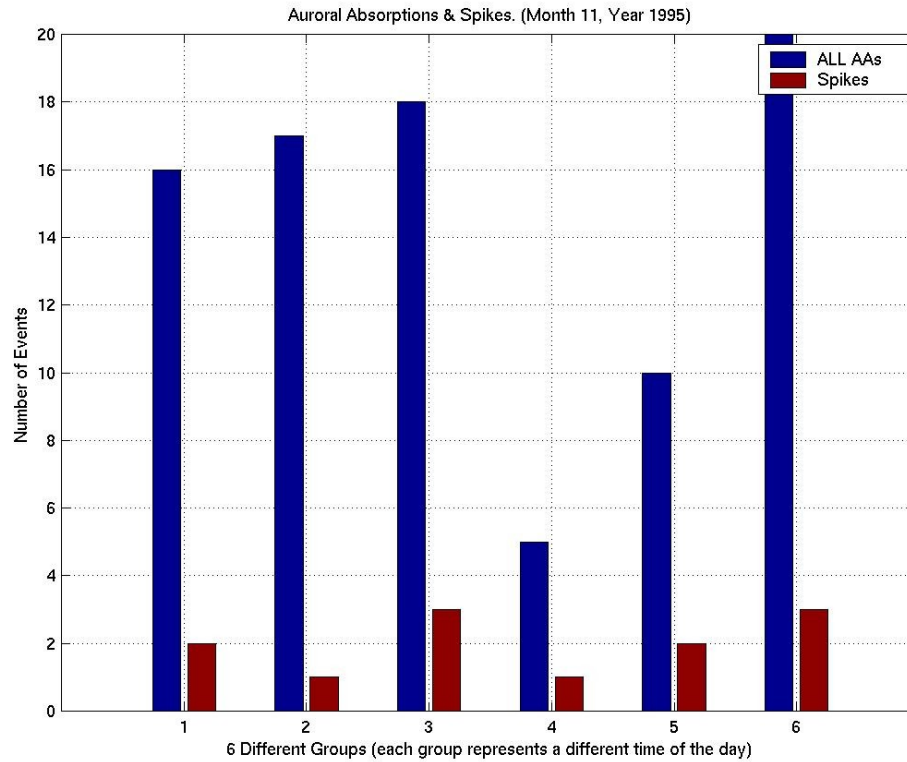


Figure 5.1.14: Statistical Analysis I, Month 11 Year 1995.

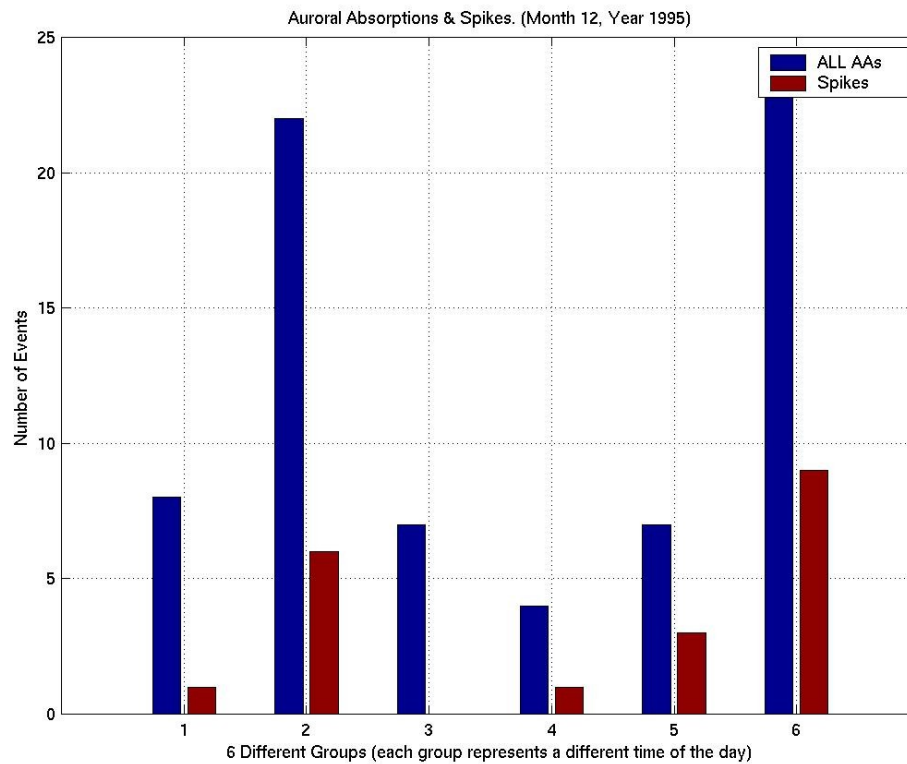


Figure 5.1.15: Statistical Analysis I, Month 12 Year 1995.

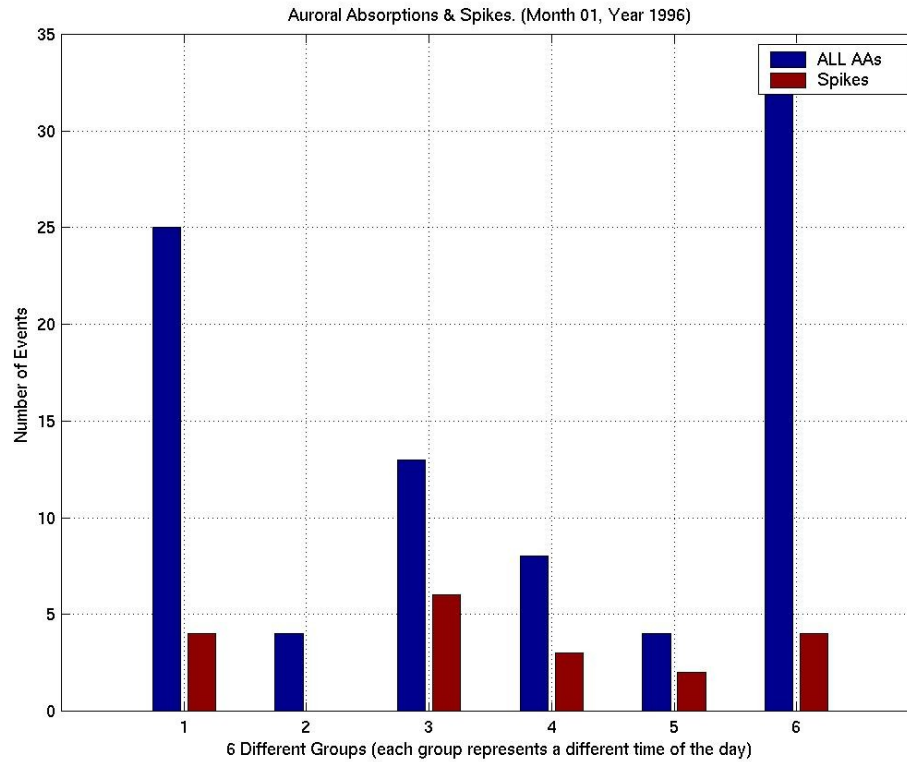


Figure 5.1.16: Statistical Analysis I, Month 01 Year 1996.

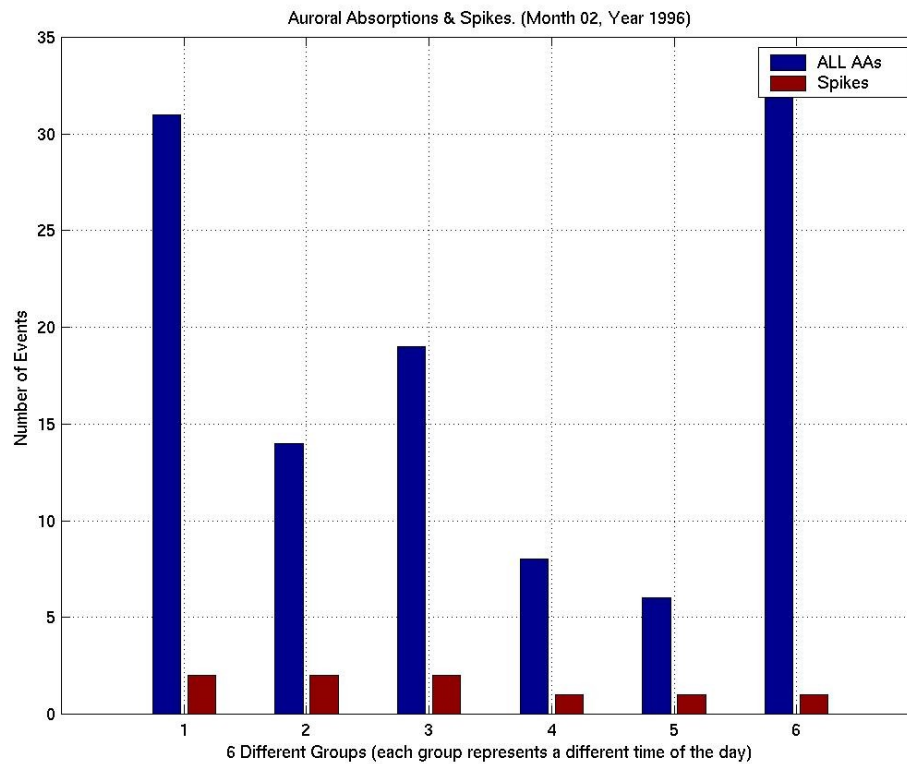


Figure 5.1.17: Statistical Analysis I, Month 02 Year 1996.

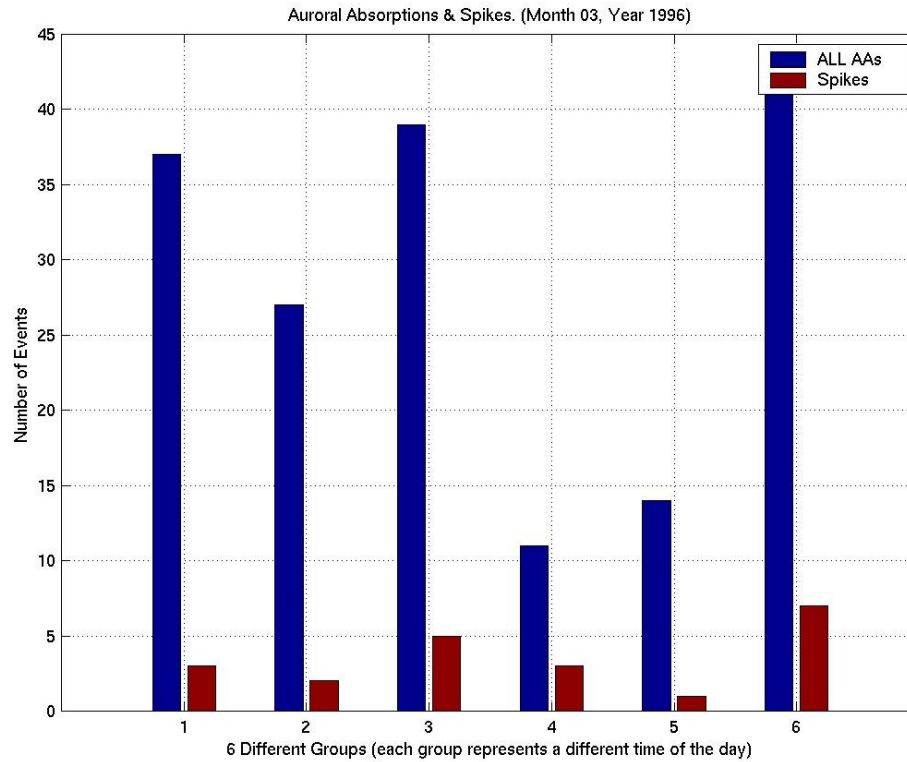


Figure 5.1.18: Statistical Analysis I, Month 03 Year 1996.

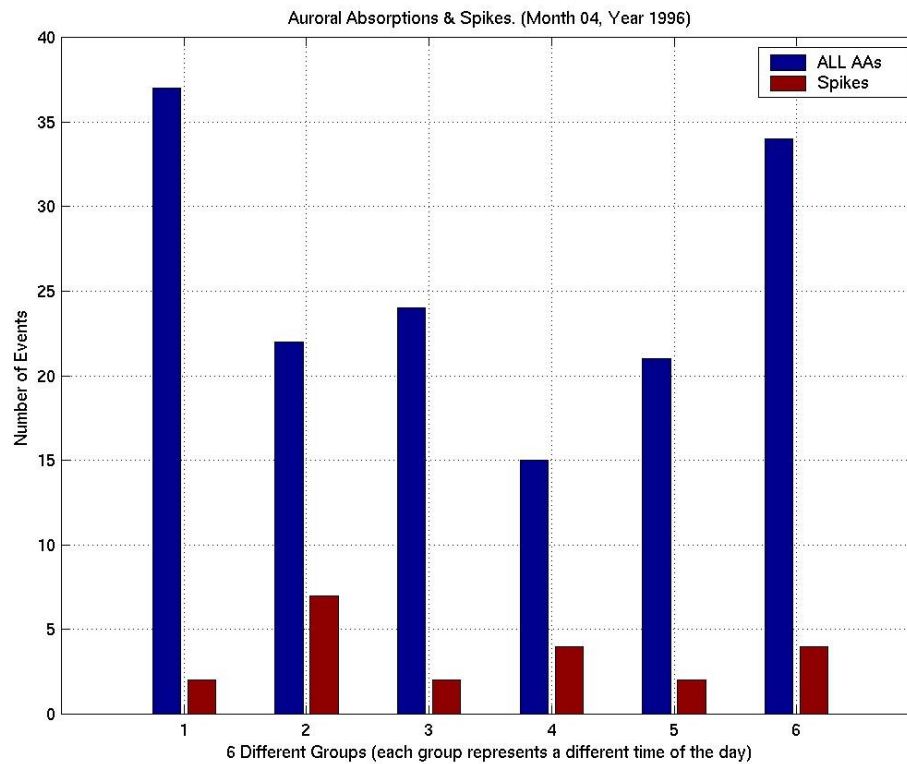


Figure 5.1.19: Statistical Analysis I, Month 04 Year 1996.

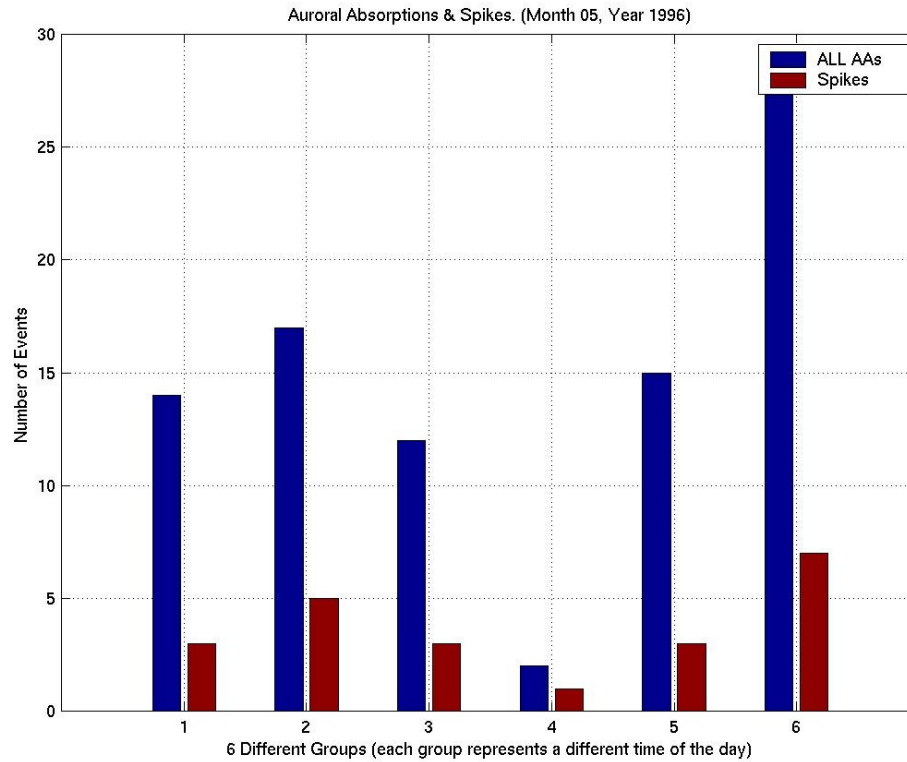


Figure 5.1.20: Statistical Analysis I, Month 05 Year 1996.

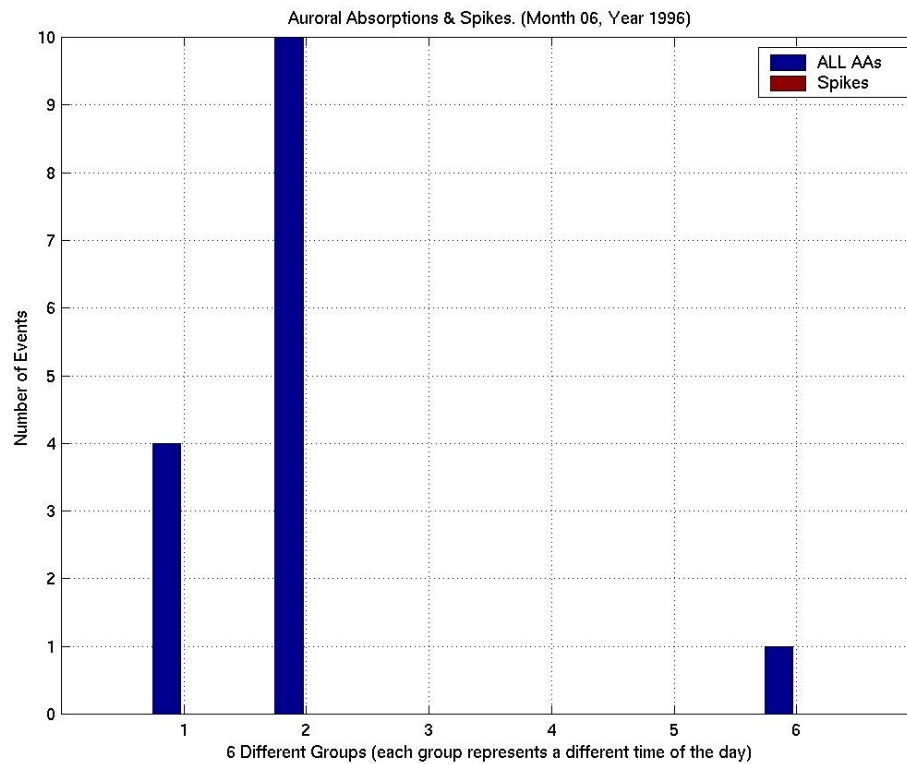


Figure 5.1.21: Statistical Analysis I, Month 06 Year 1996.

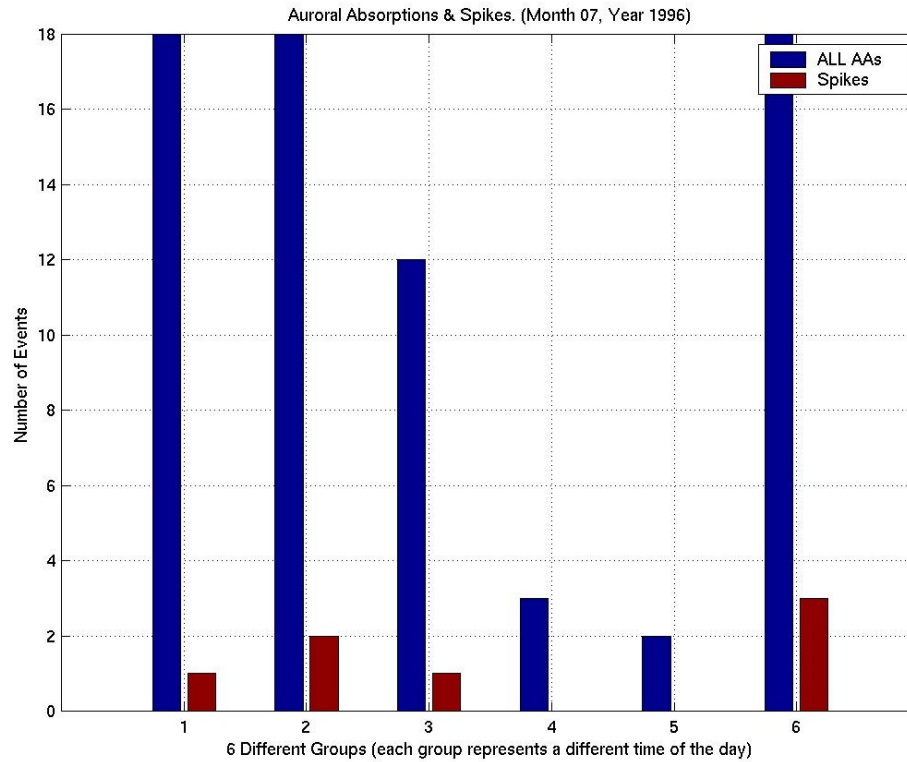


Figure 5.1.22: Statistical Analysis I, Month 07 Year 1996.

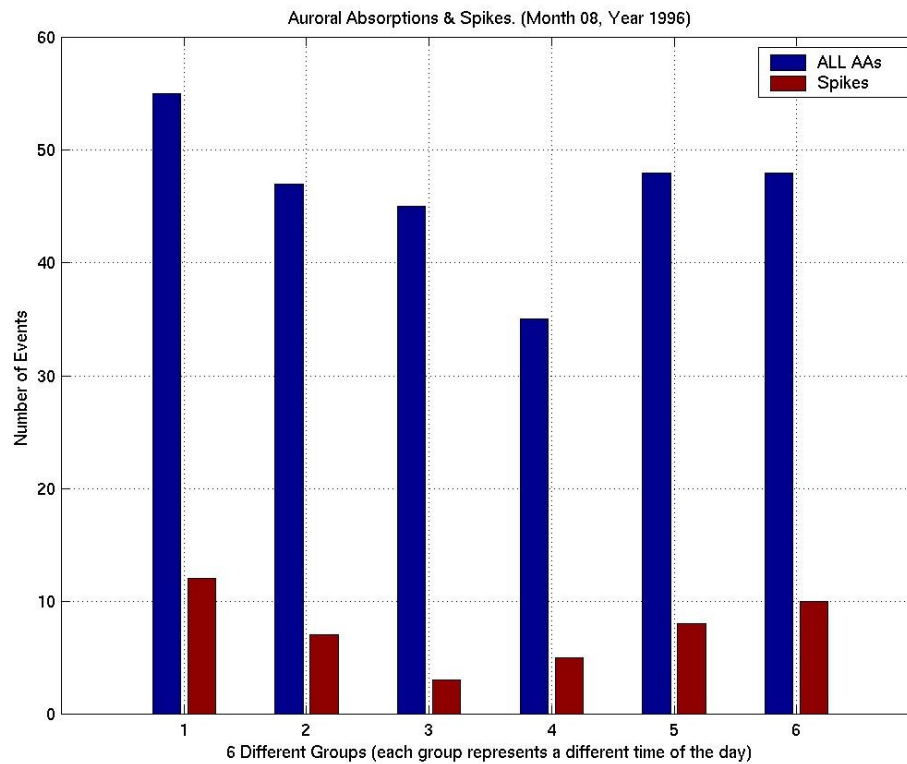


Figure 5.1.23: Statistical Analysis I, Month 08 Year 1996.

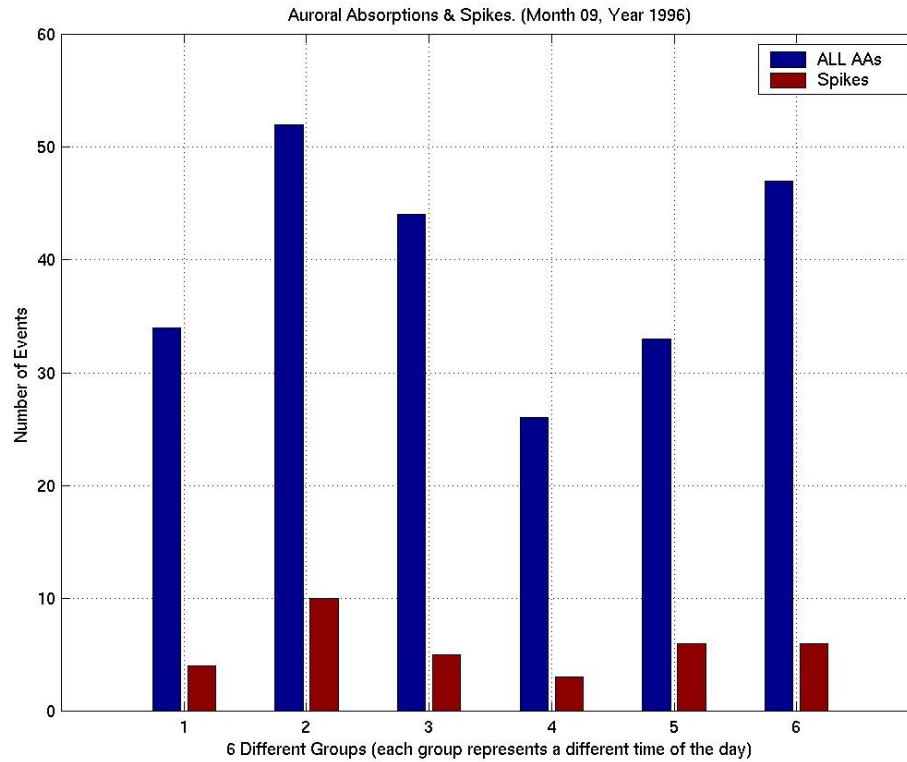


Figure 5.1.24: Statistical Analysis I, Month 09 Year 1996.

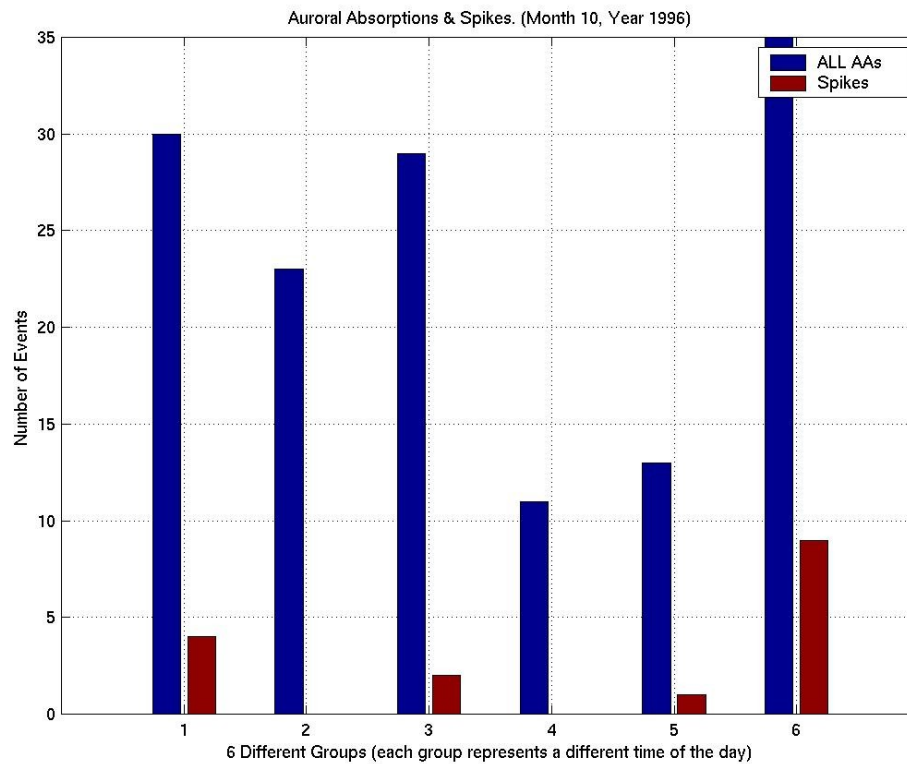


Figure 5.1.25: Statistical Analysis I, Month 10 Year 1996.

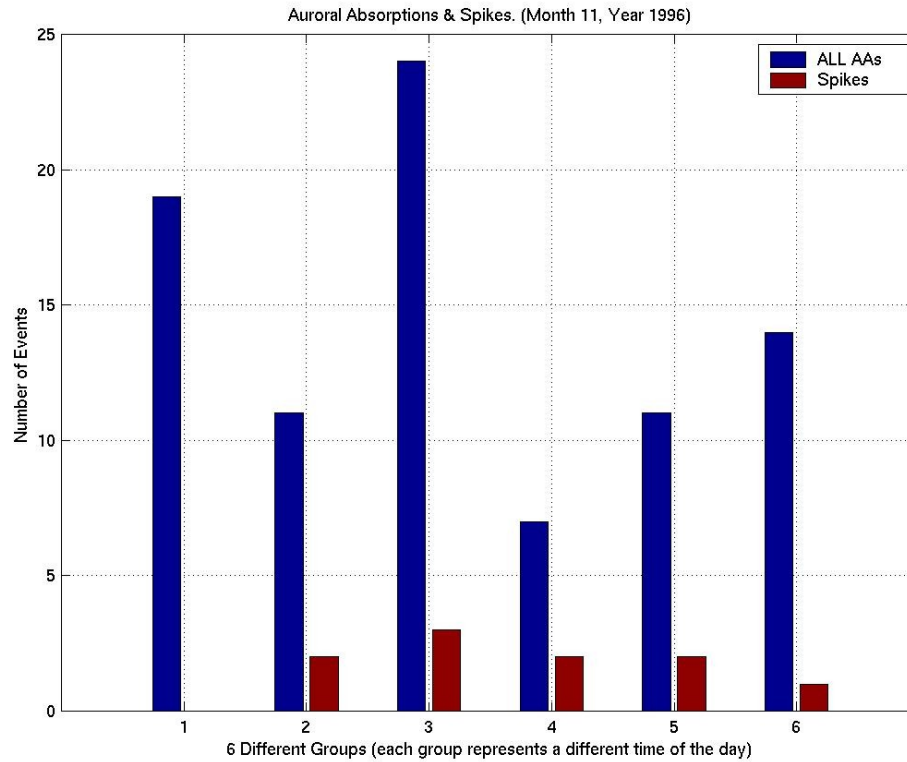


Figure 5.1.26: Statistical Analysis I, Month 11 Year 1996.

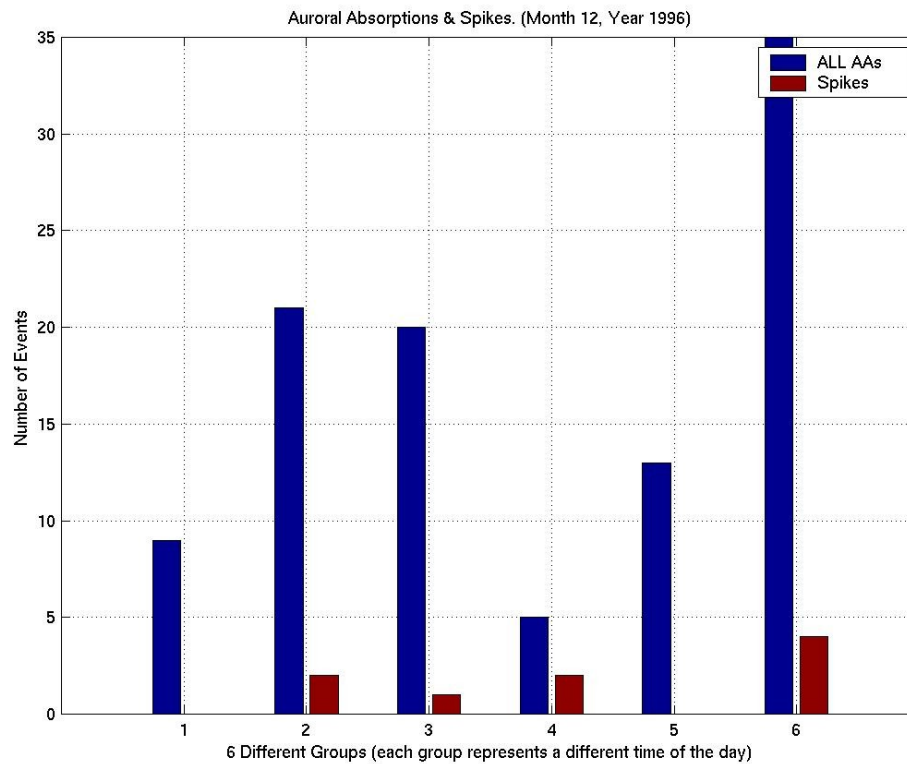


Figure 5.1.27: Statistical Analysis I, Month 12 Year 1996.

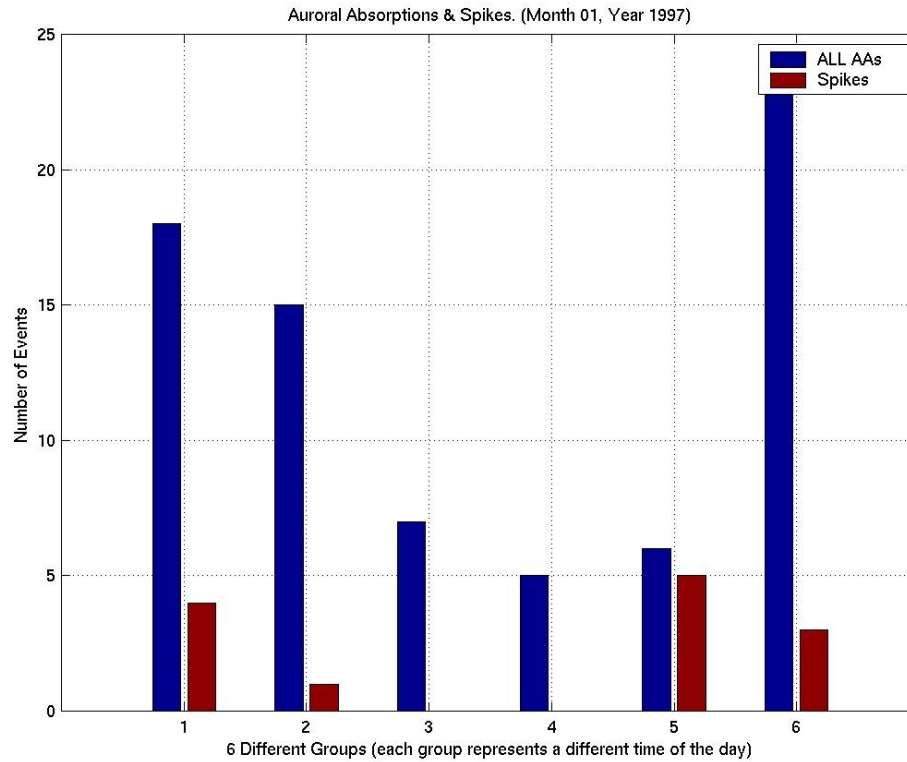


Figure 5.1.28: Statistical Analysis I, Month 01 Year 1997.

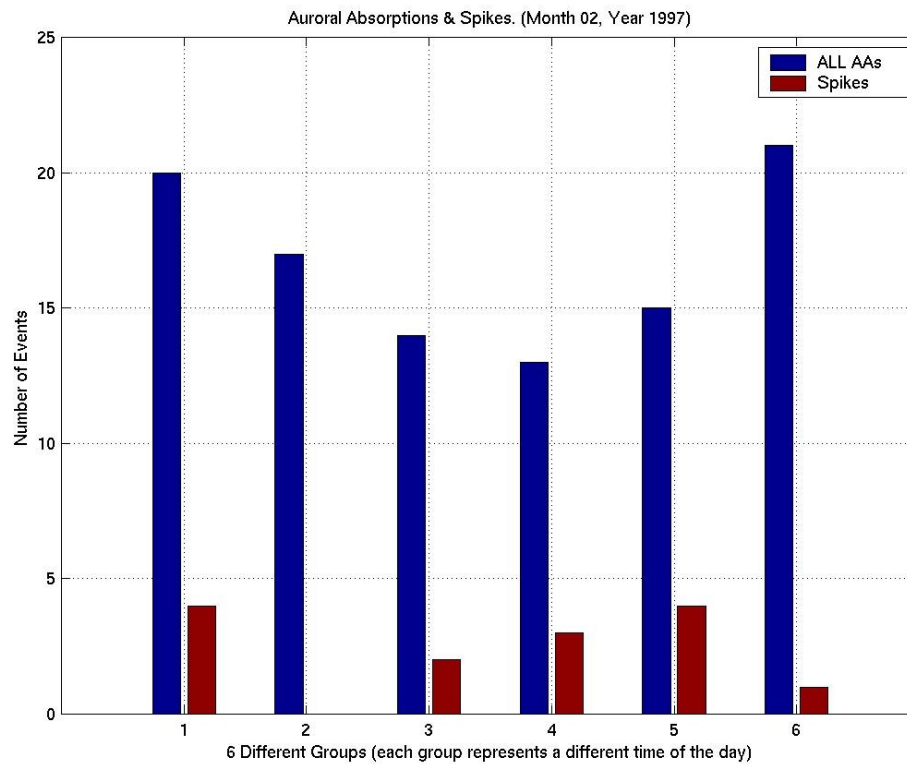


Figure 5.1.29: Statistical Analysis I, Month 02 Year 1997.

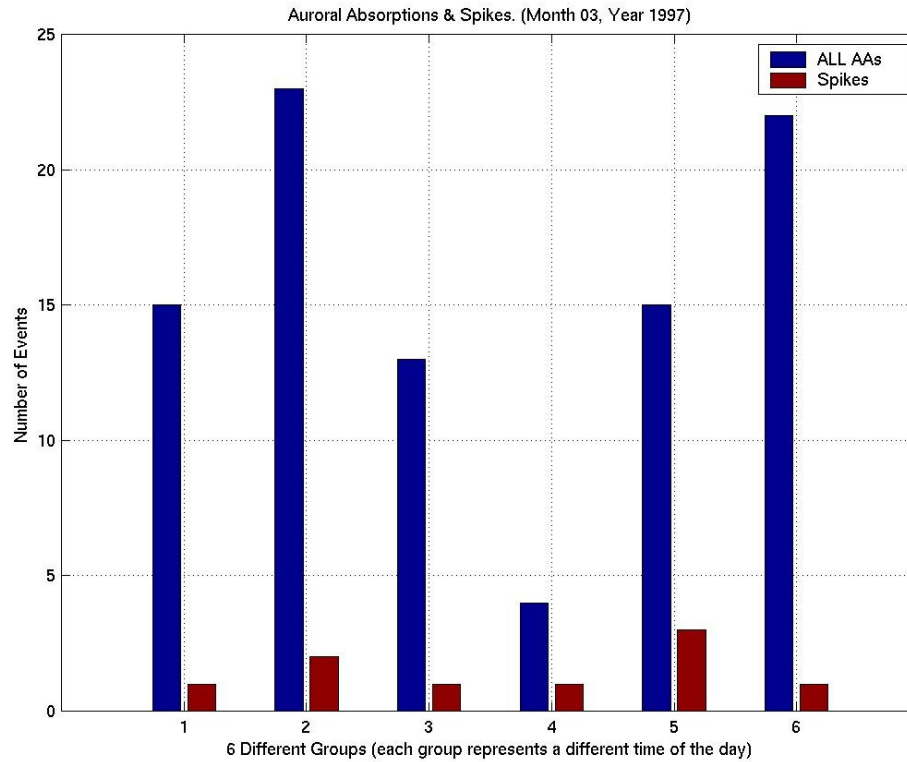


Figure 5.1.30: Statistical Analysis I, Month 03 Year 1997.

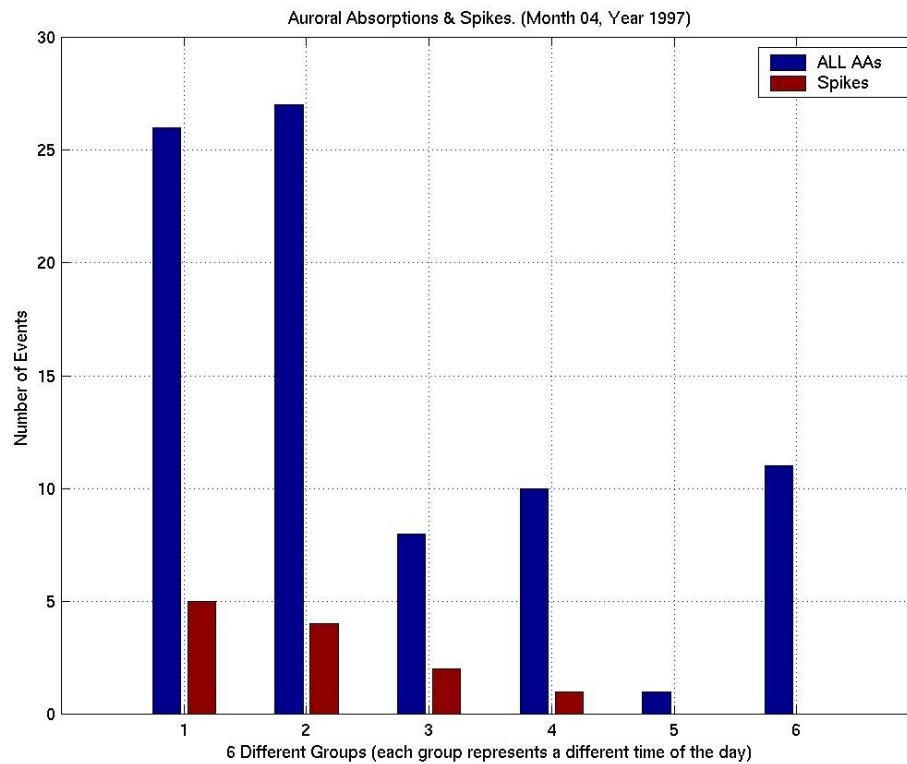


Figure 5.1.31: Statistical Analysis I, Month 04 Year 1997.

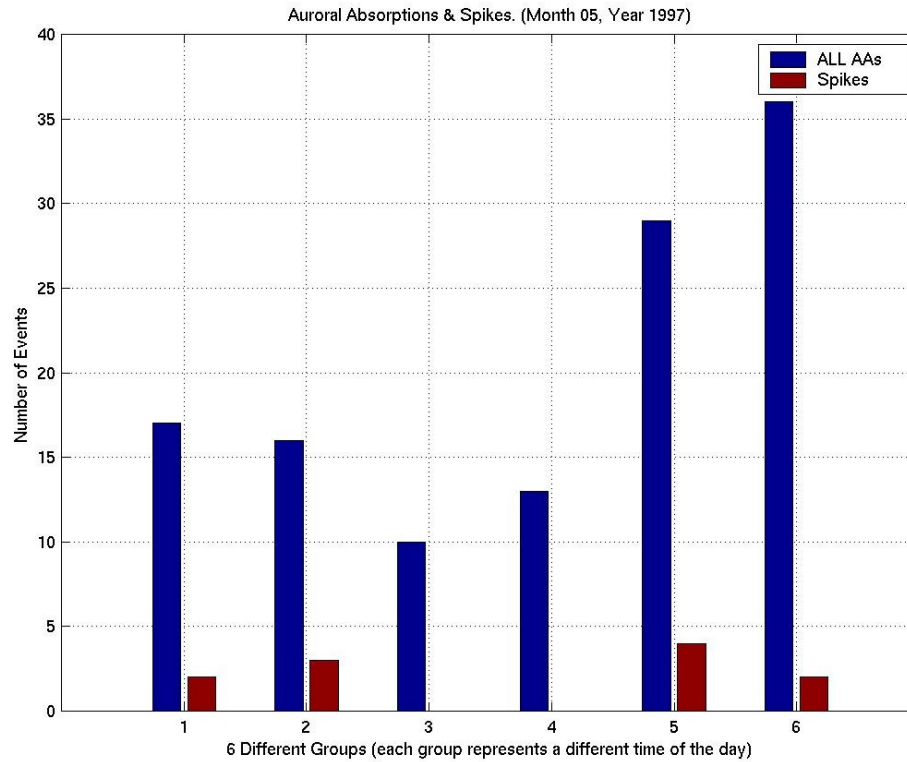


Figure 5.1.32: Statistical Analysis I, Month 05 Year 1997.

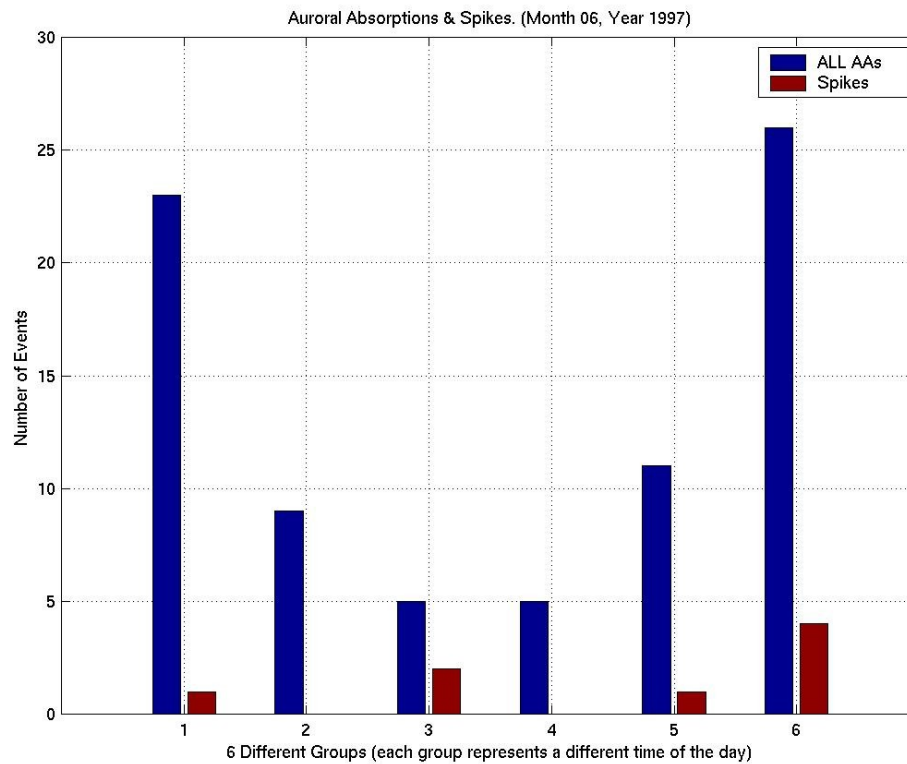


Figure 5.1.33: Statistical Analysis I, Month 06 Year 1997.

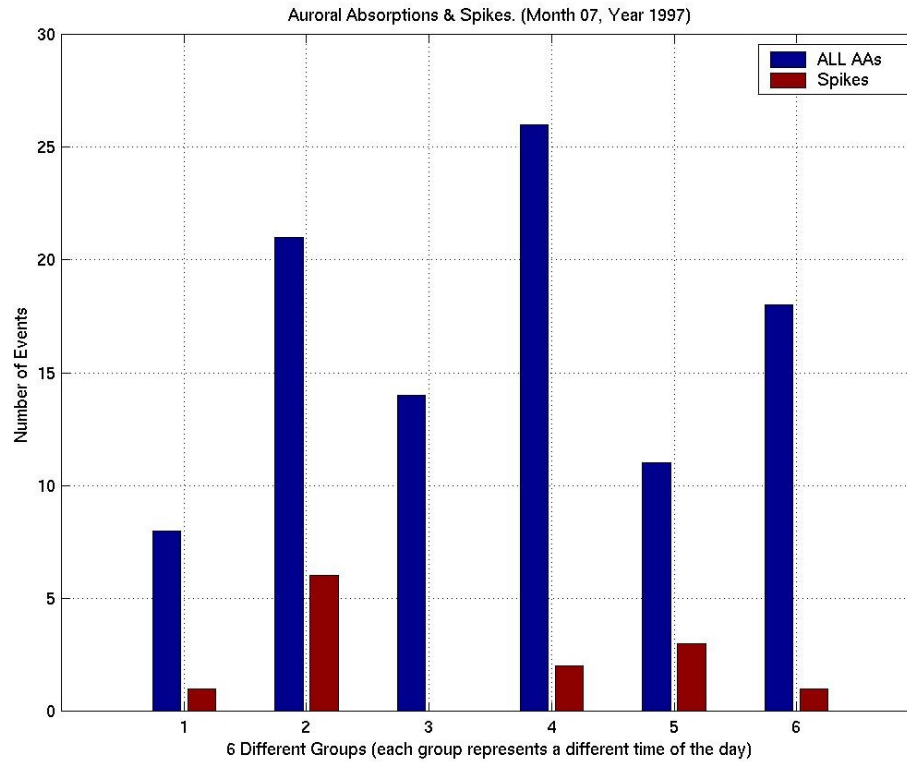


Figure 5.1.34: Statistical Analysis I, Month 07 Year 1997.

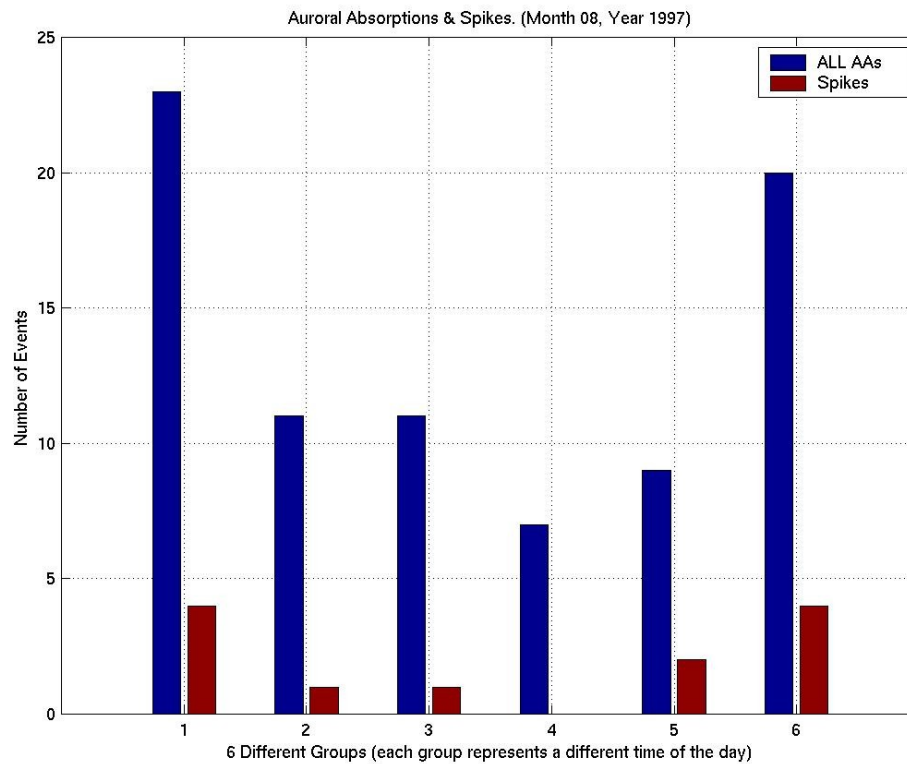


Figure 5.1.35: Statistical Analysis I, Month 08 Year 1997.

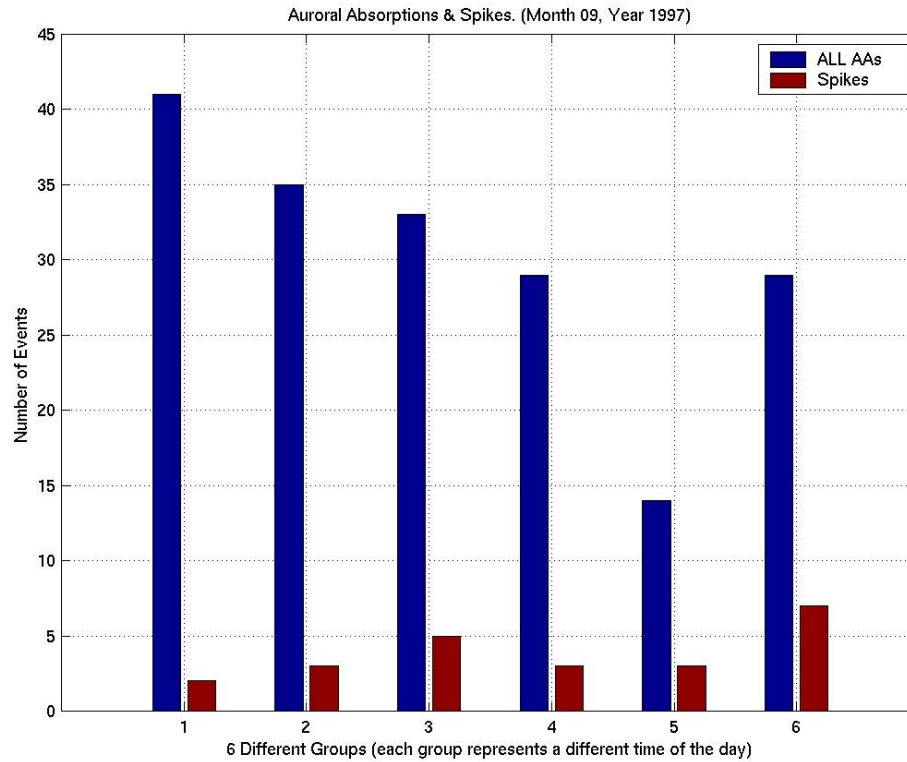


Figure 5.1.36: Statistical Analysis I, Month 09 Year 1997.

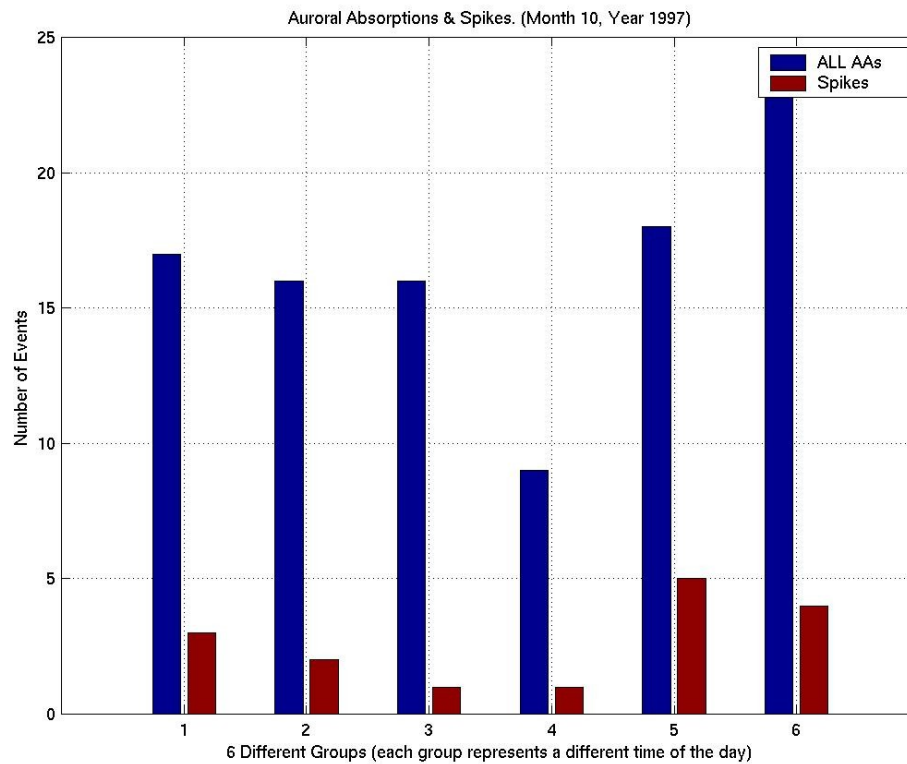


Figure 5.1.37: Statistical Analysis I, Month 10 Year 1997.

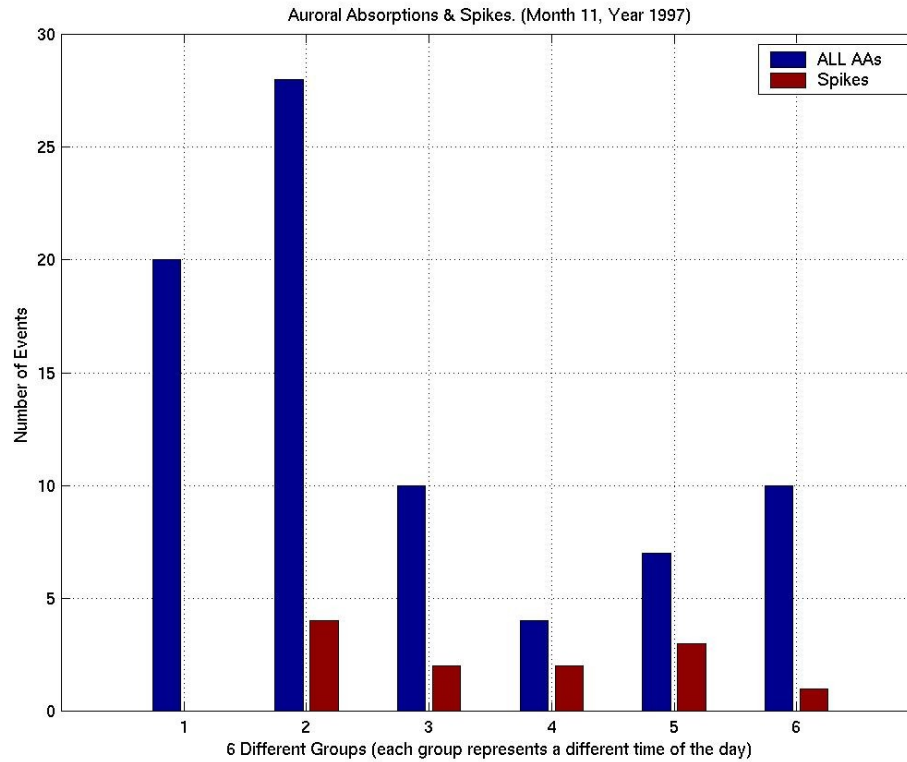


Figure 5.1.38: Statistical Analysis I, Month 11 Year 1997.

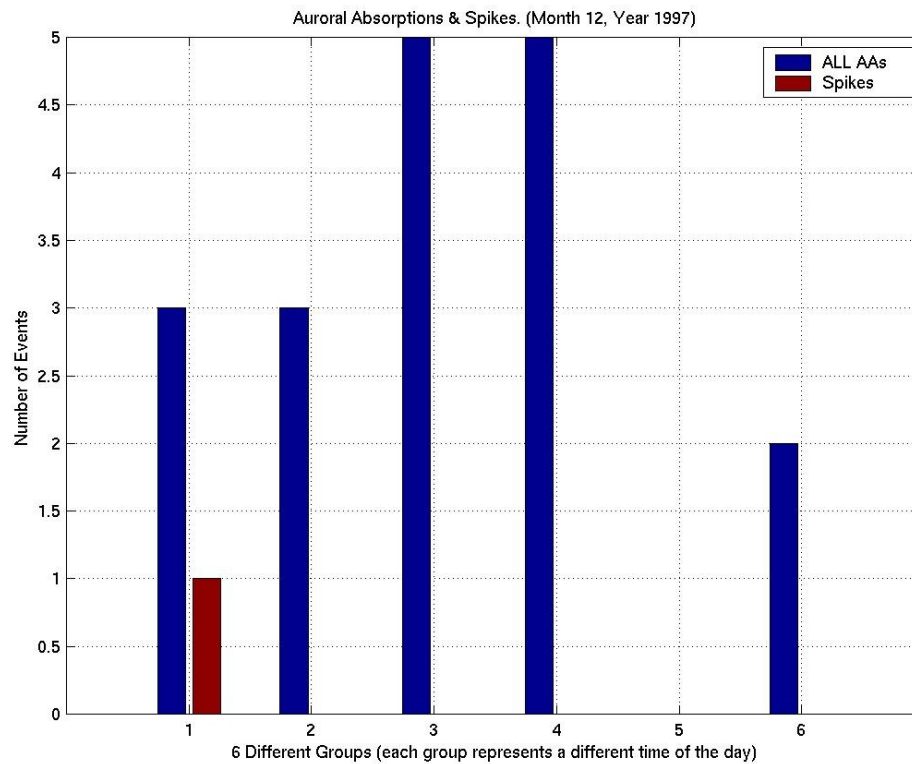


Figure 5.1.39: Statistical Analysis I, Month 12 Year 1997.

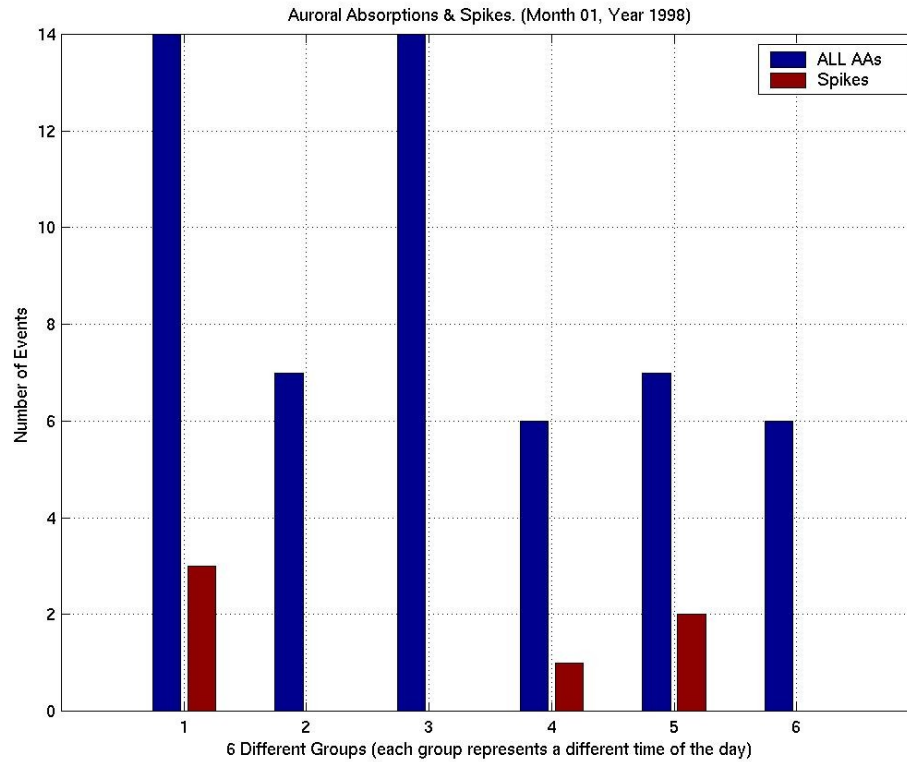


Figure 5.1.40: Statistical Analysis I, Month 01 Year 1998.

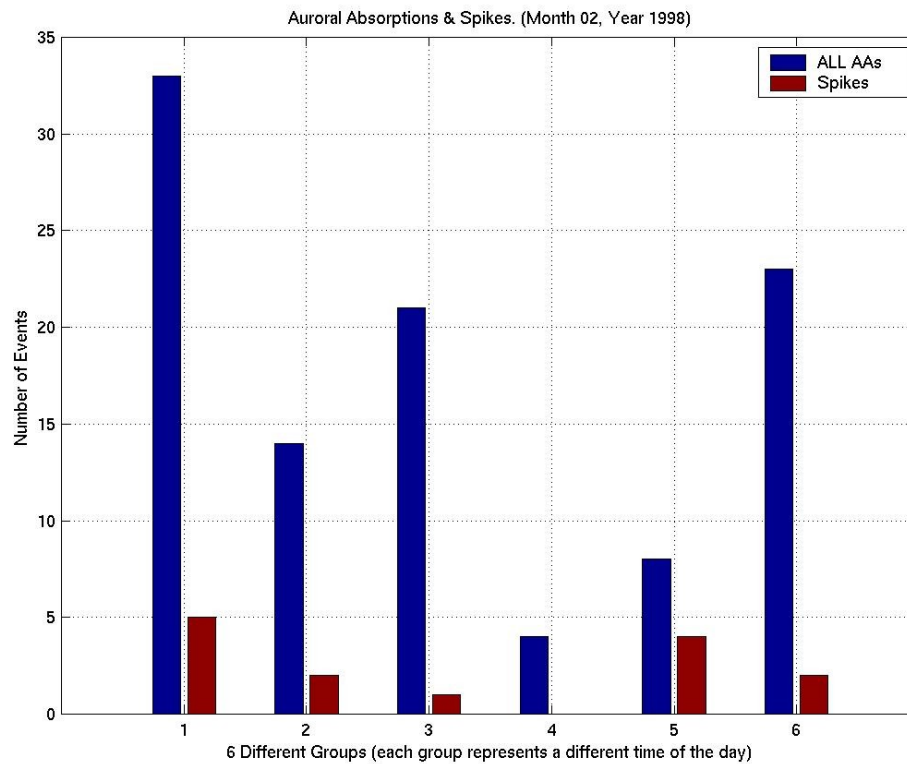


Figure 5.1.41: Statistical Analysis I, Month 02 Year 1998.

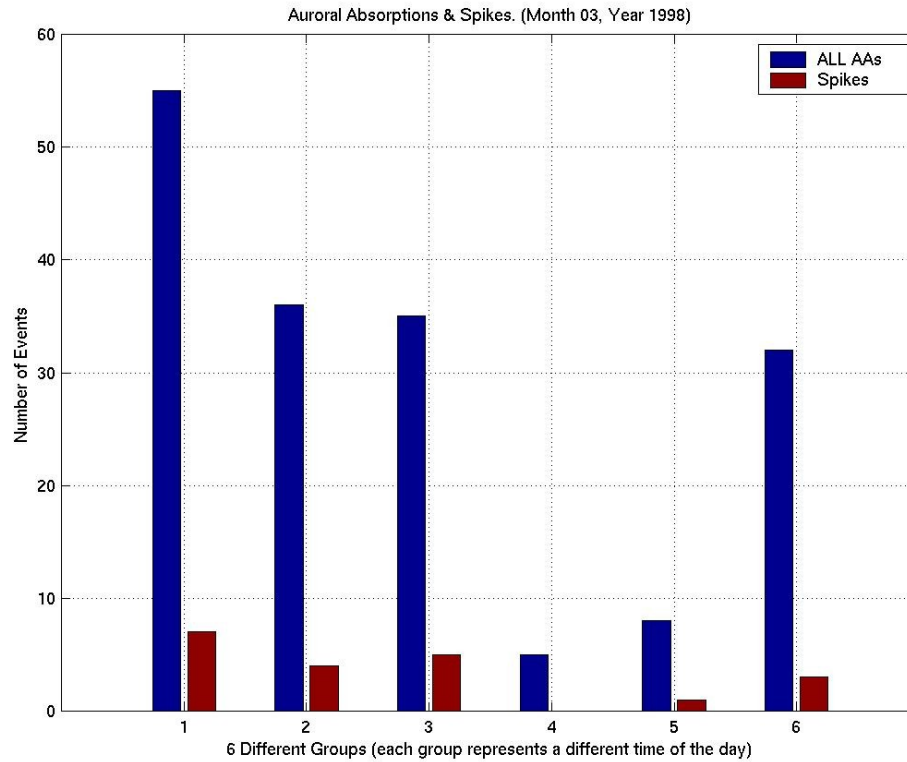


Figure 5.1.42: Statistical Analysis I, Month 03 Year 1998.

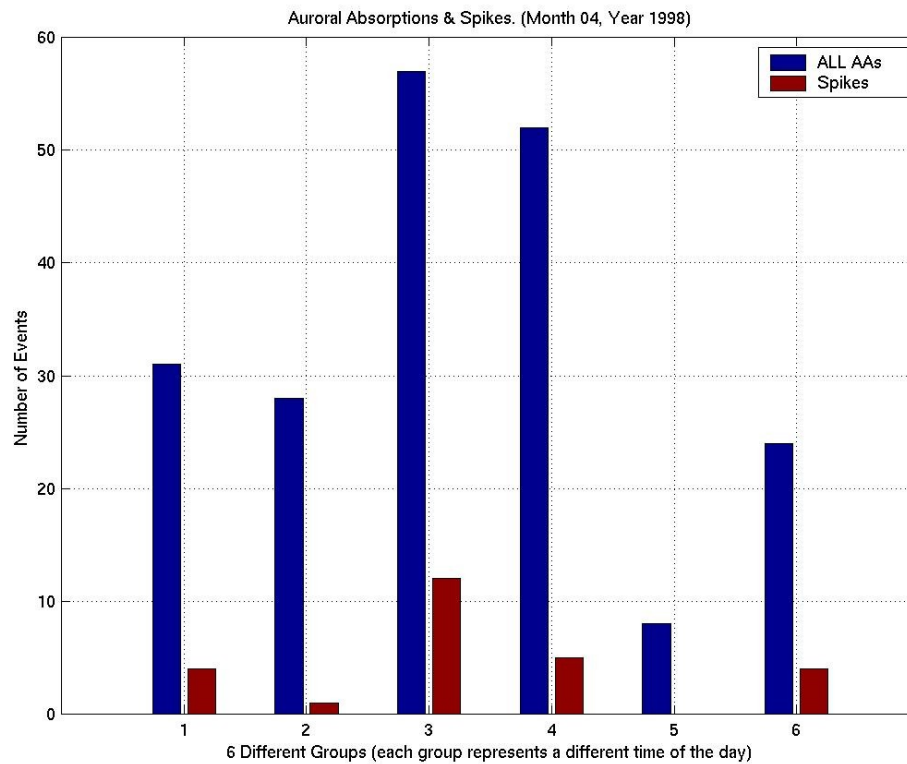


Figure 5.1.43: Statistical Analysis I, Month 04 Year 1998.

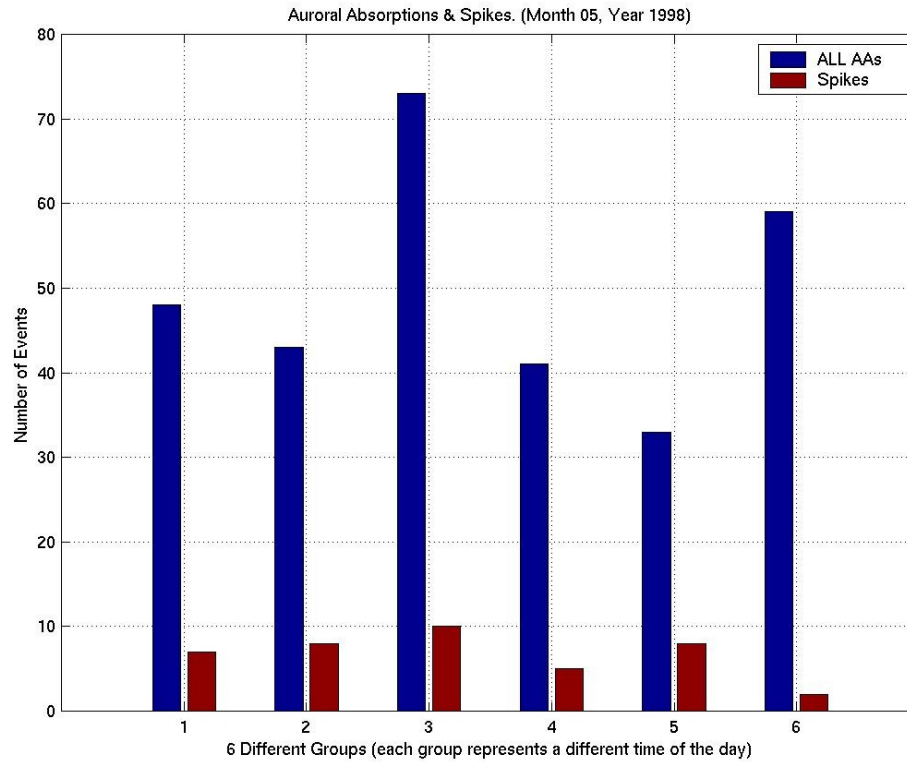


Figure 5.1.44: Statistical Analysis I, Month 05 Year 1998.

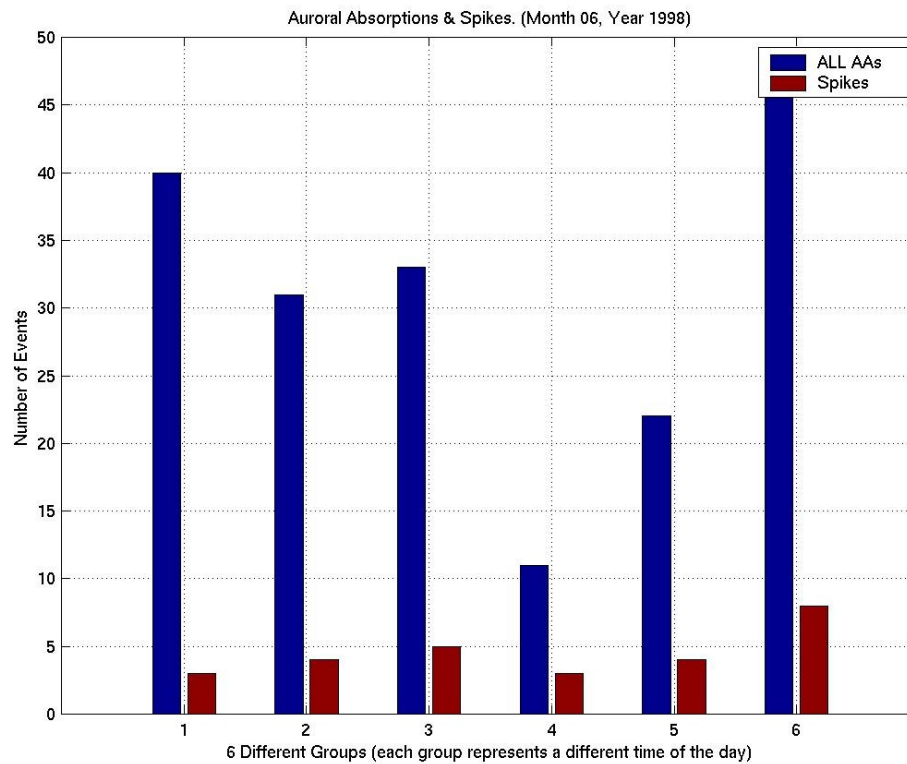


Figure 5.1.45: Statistical Analysis I, Month 06 Year 1998.

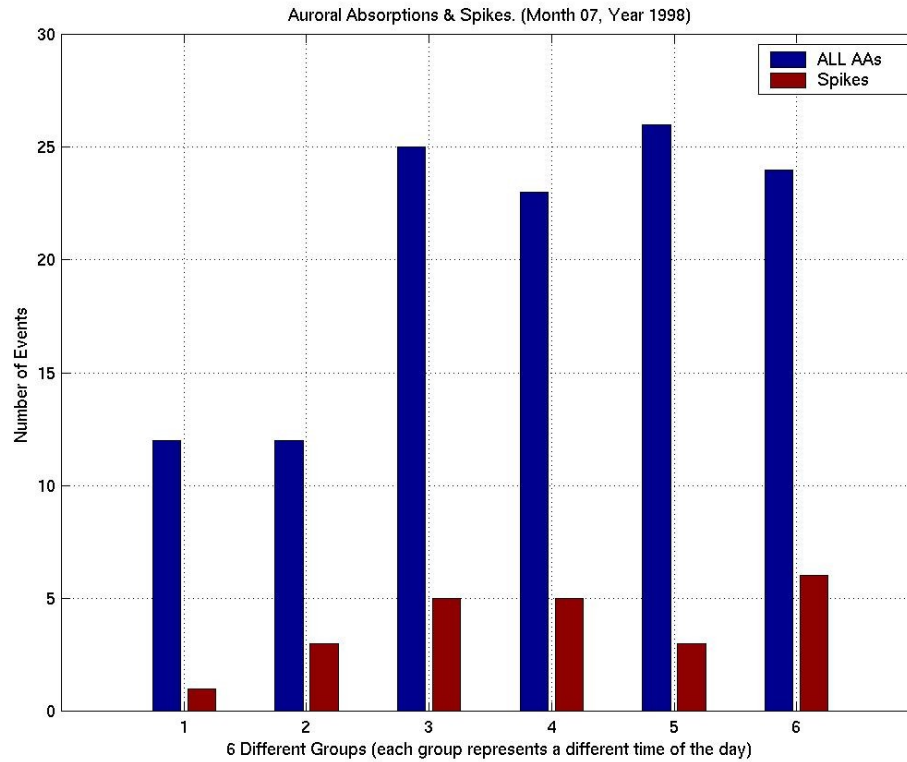


Figure 5.1.46: Statistical Analysis I, Month 07 Year 1998.

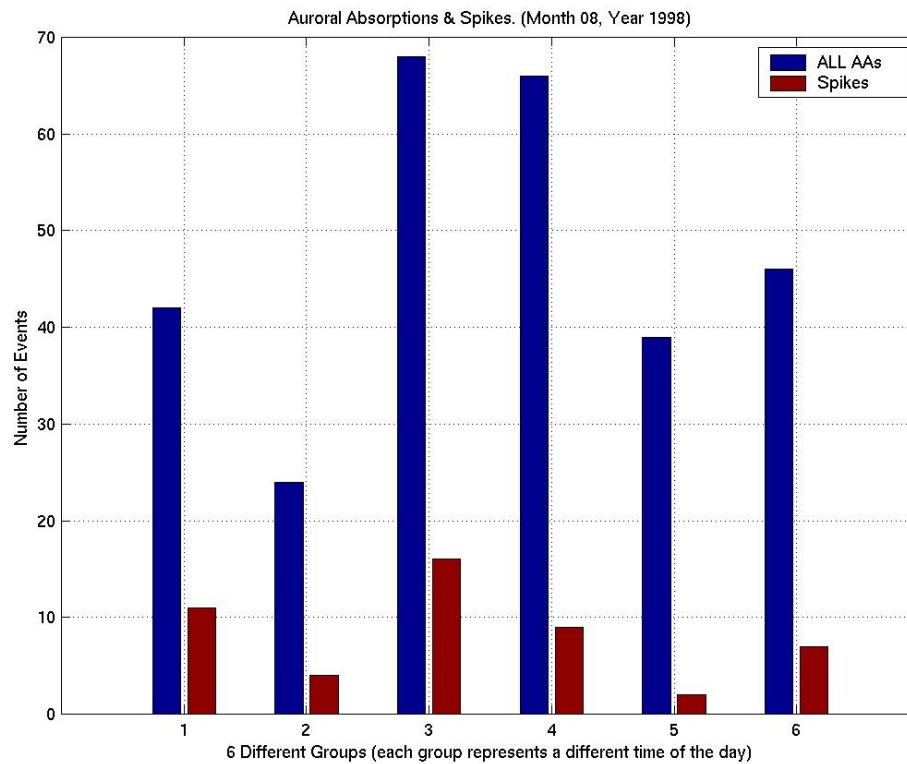


Figure 5.1.47: Statistical Analysis I, Month 08 Year 1998.

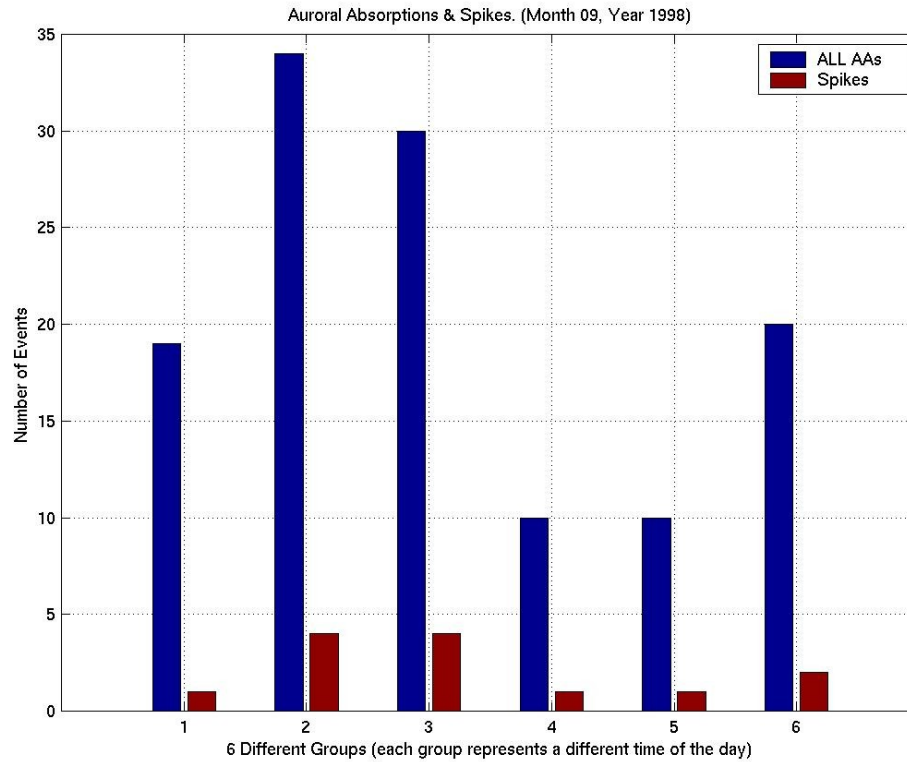


Figure 5.1.48: Statistical Analysis I, Month 09 Year 1998.

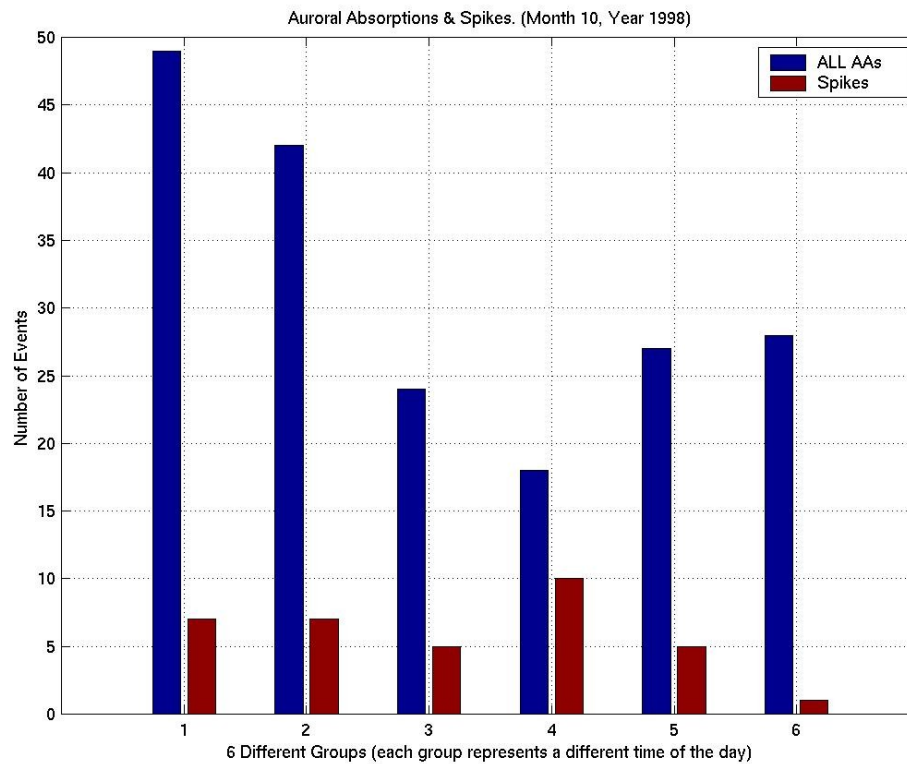


Figure 5.1.49: Statistical Analysis I, Month 10 Year 1998.

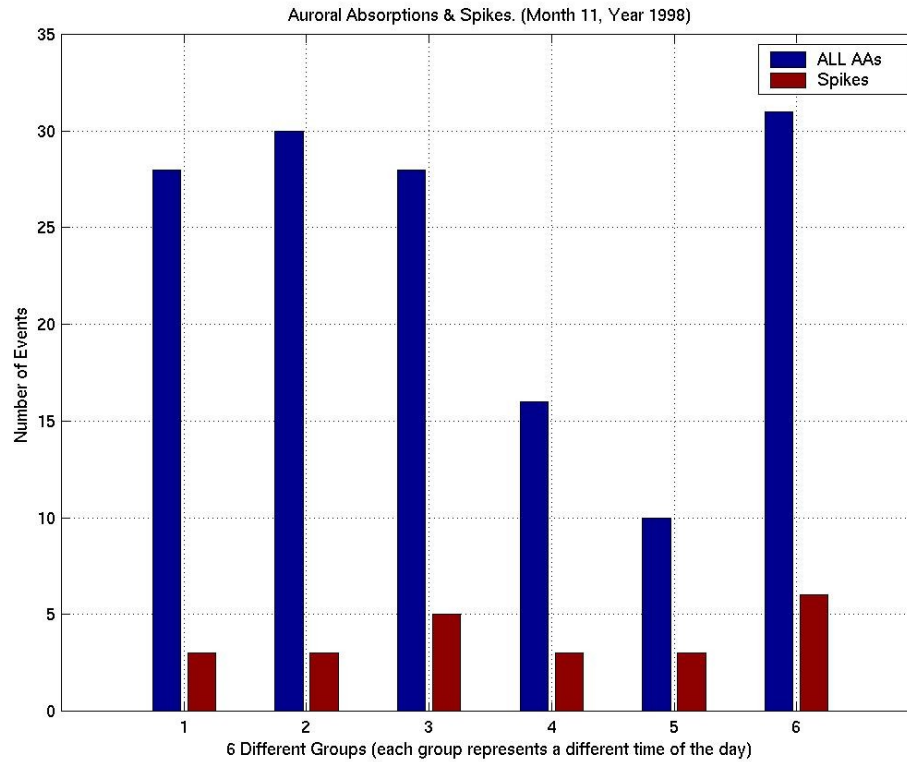


Figure 5.1.50: Statistical Analysis I, Month 11 Year 1998.

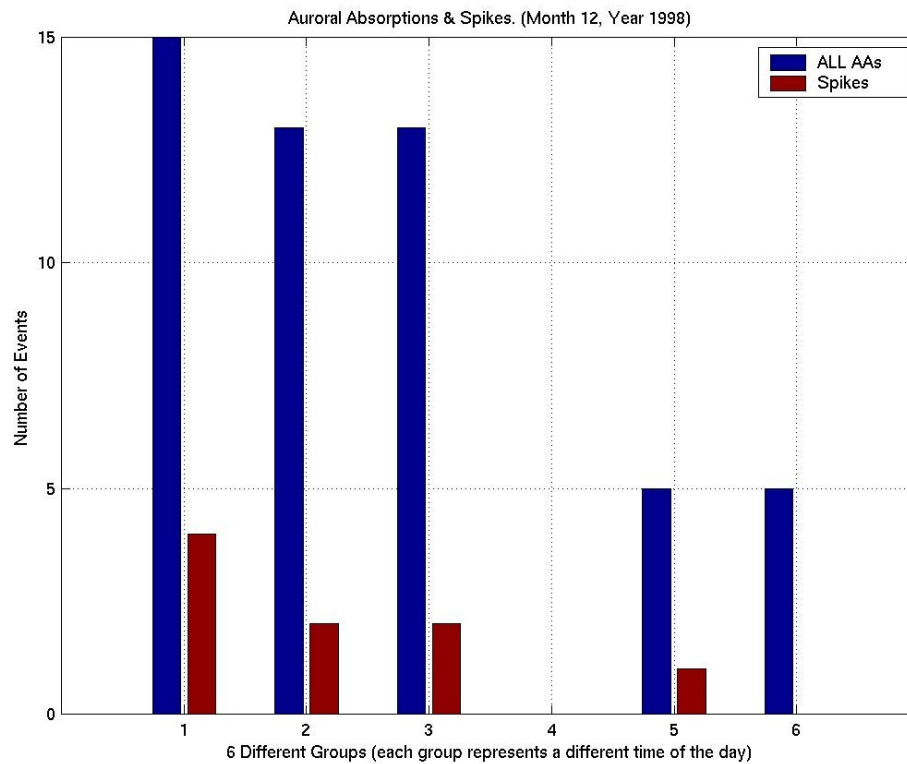


Figure 5.1.51: Statistical Analysis I, Month 12 Year 1998.

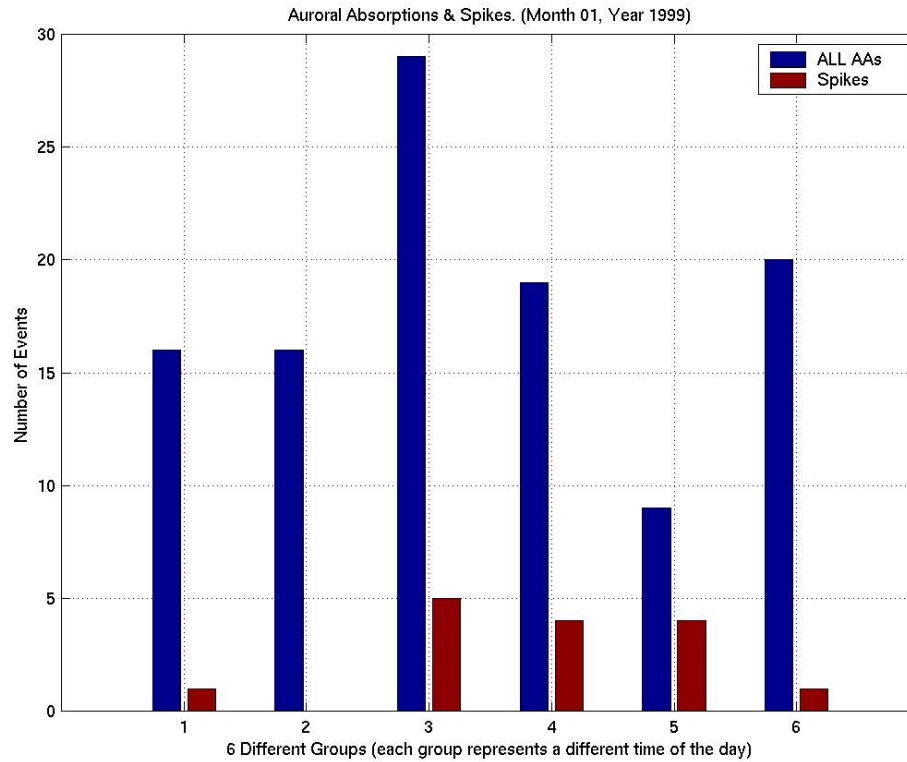


Figure 5.1.52: Statistical Analysis I, Month 01 Year 1999.

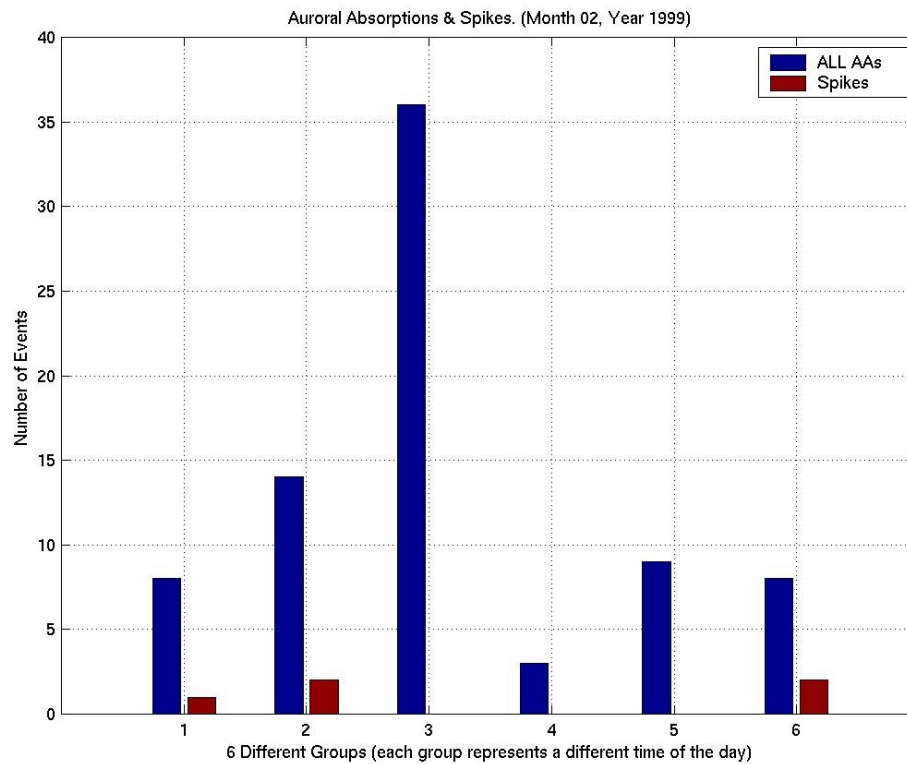


Figure 5.1.53: Statistical Analysis I, Month 02 Year 1999.

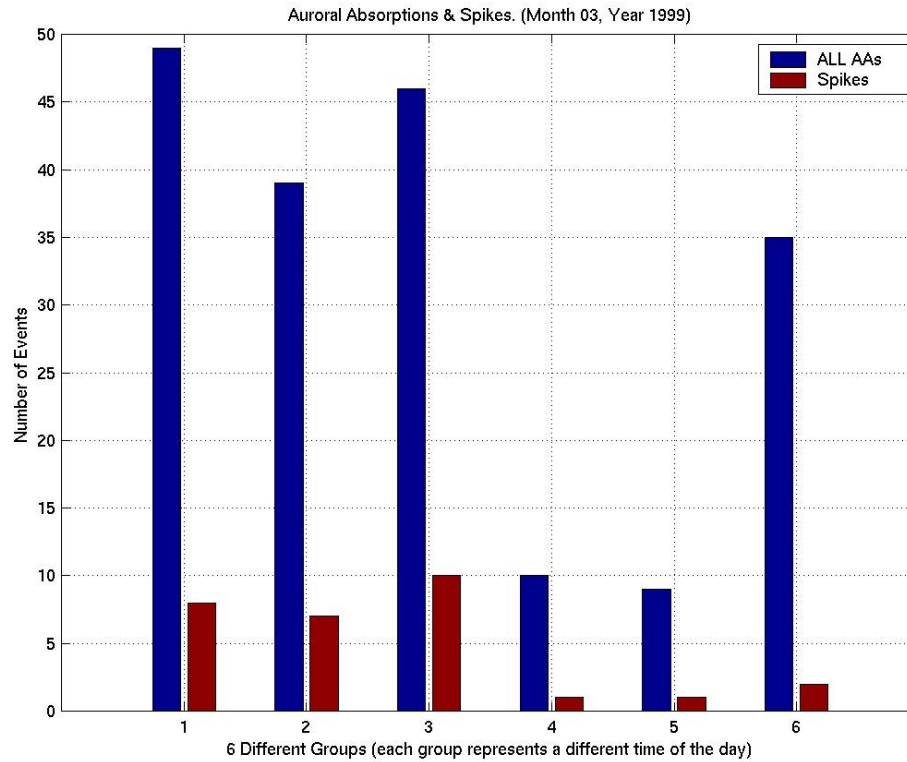


Figure 5.1.54: Statistical Analysis I, Month 03 Year 1999.

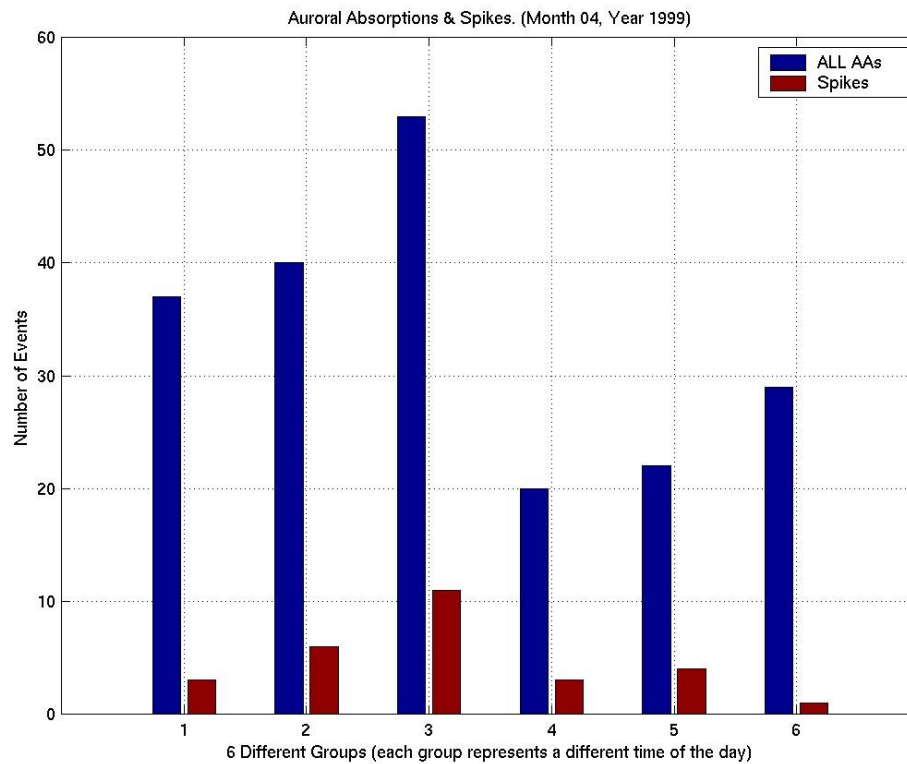


Figure 5.1.55: Statistical Analysis I, Month 04 Year 1999.

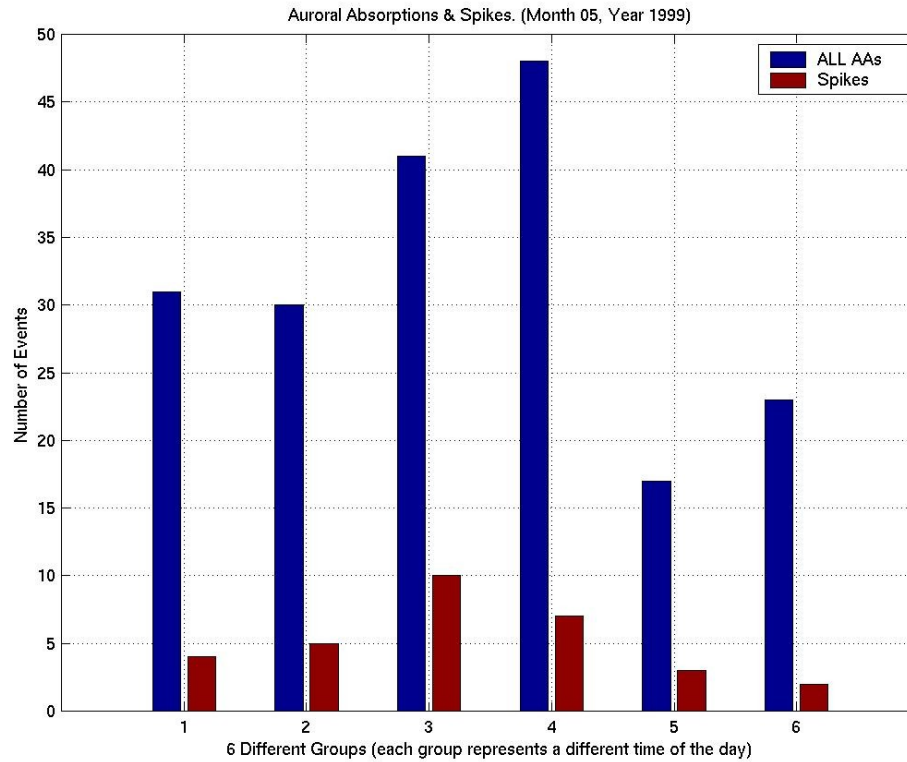


Figure 5.1.56: Statistical Analysis I, Month 05 Year 1999.

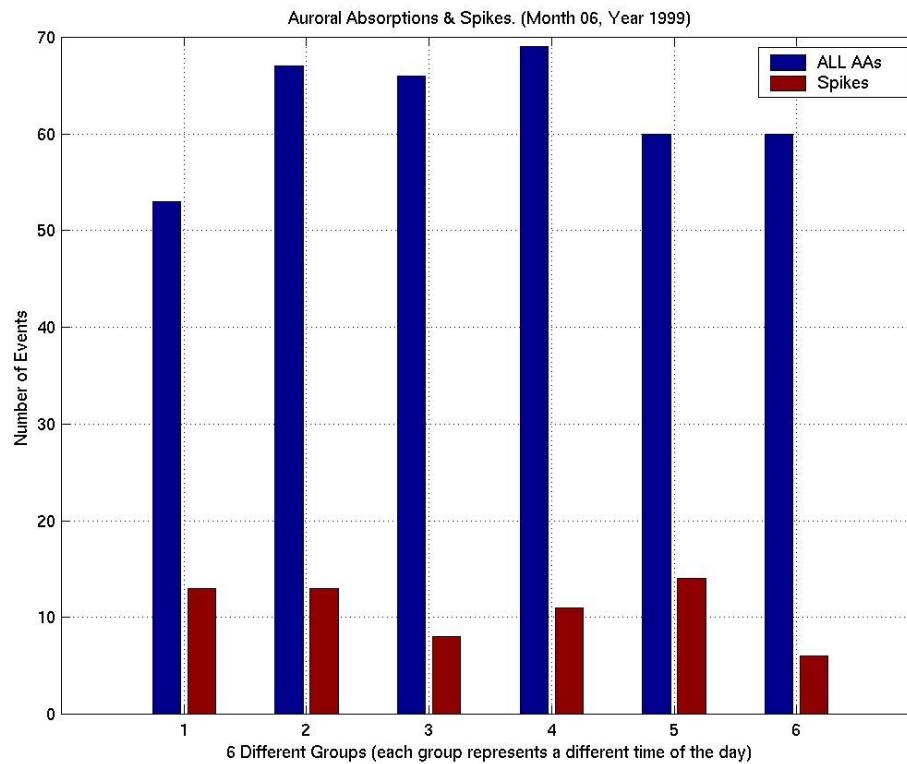


Figure 5.1.57: Statistical Analysis I, Month 06 Year 1999.

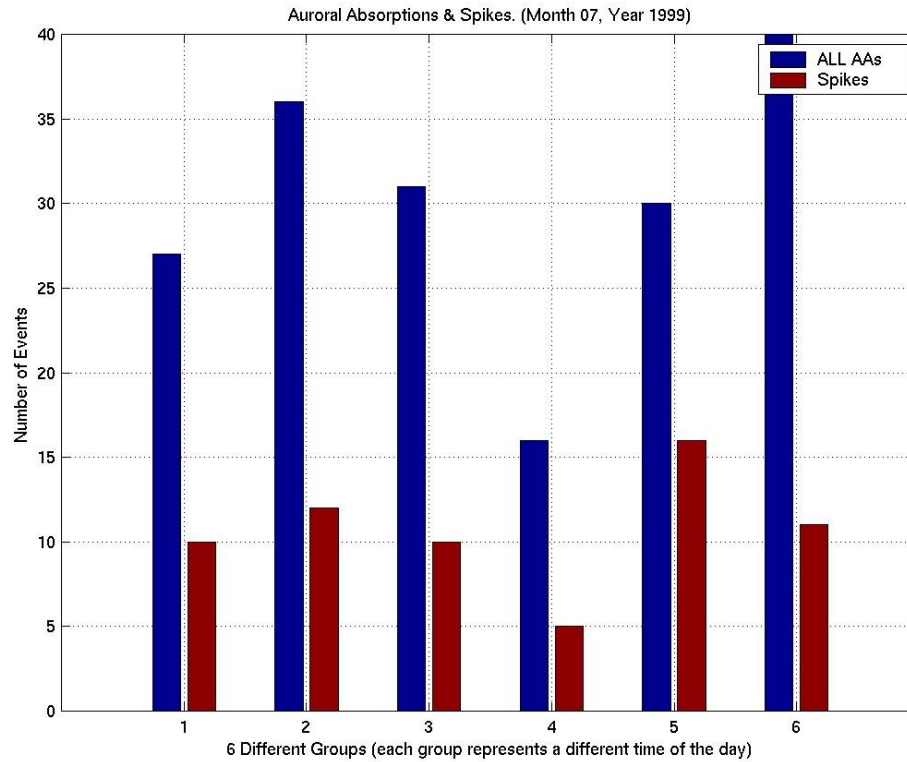


Figure 5.1.58: Statistical Analysis I, Month 07 Year 1999.

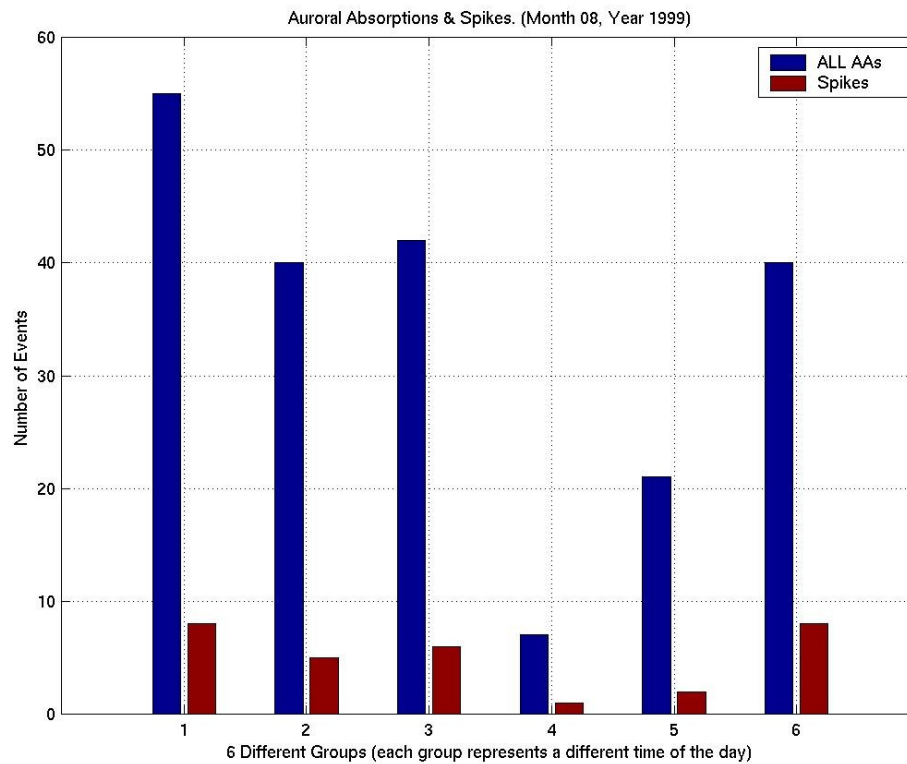


Figure 5.1.59: Statistical Analysis I, Month 08 Year 1999.

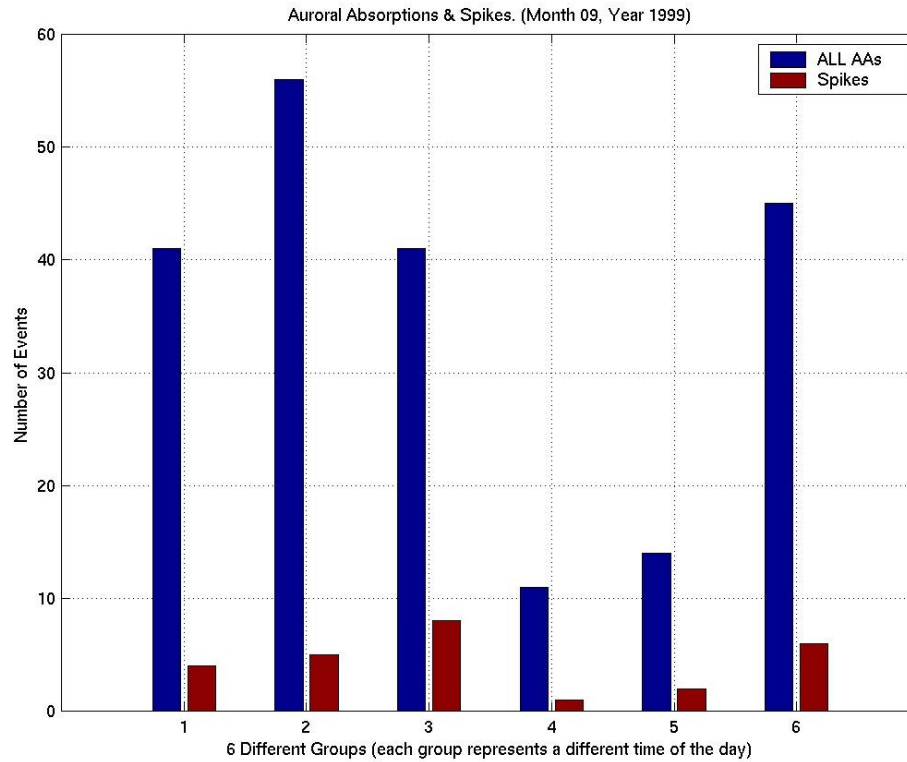


Figure 5.1.60: Statistical Analysis I, Month 09 Year 1999.

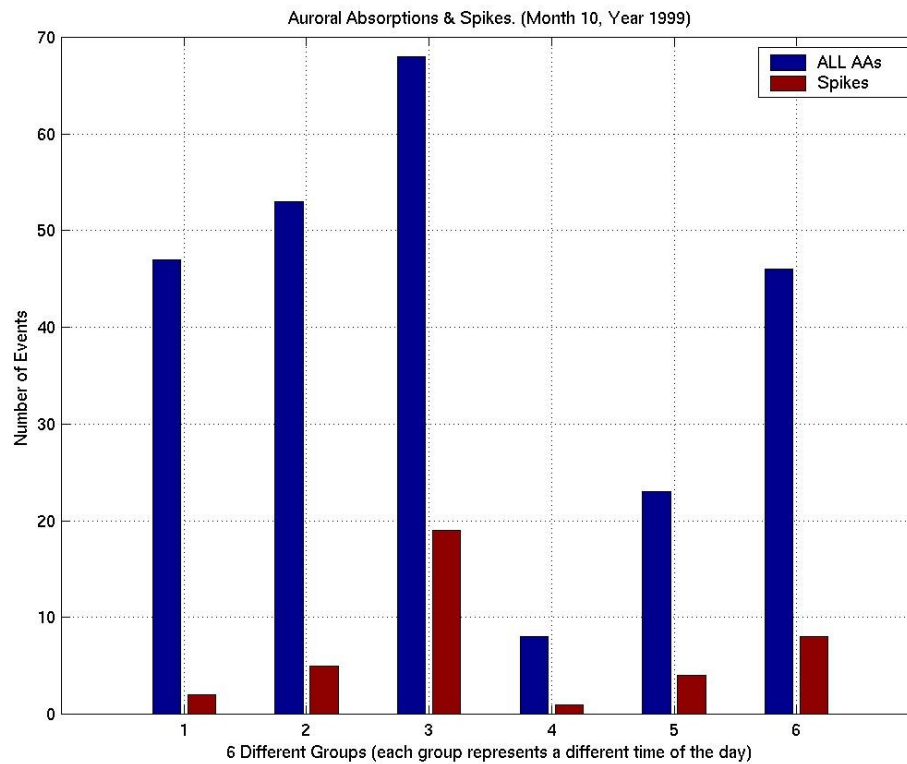


Figure 5.1.61: Statistical Analysis I, Month 10 Year 1999.

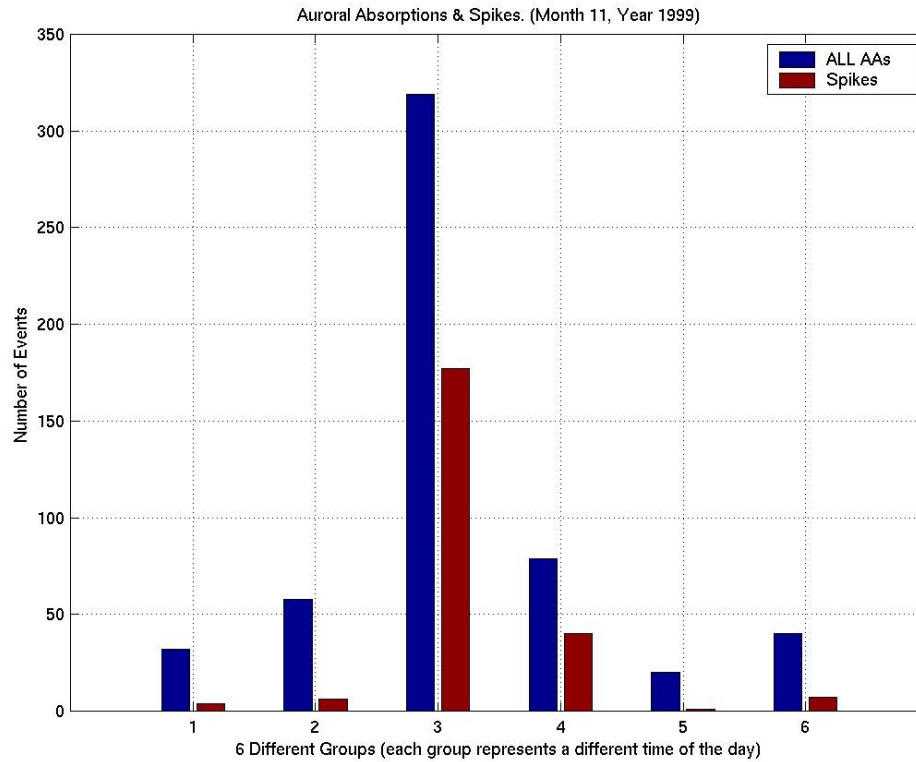


Figure 5.1.62: Statistical Analysis I, Month 11 Year 1999.

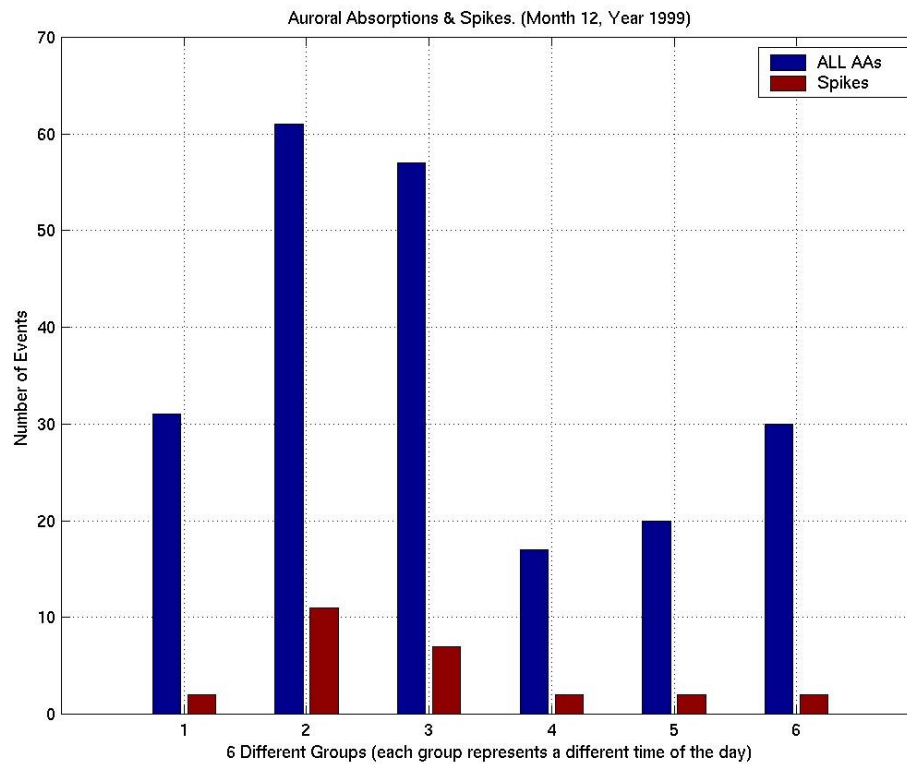


Figure 5.1.63: Statistical Analysis I, Month 12 Year 1999.

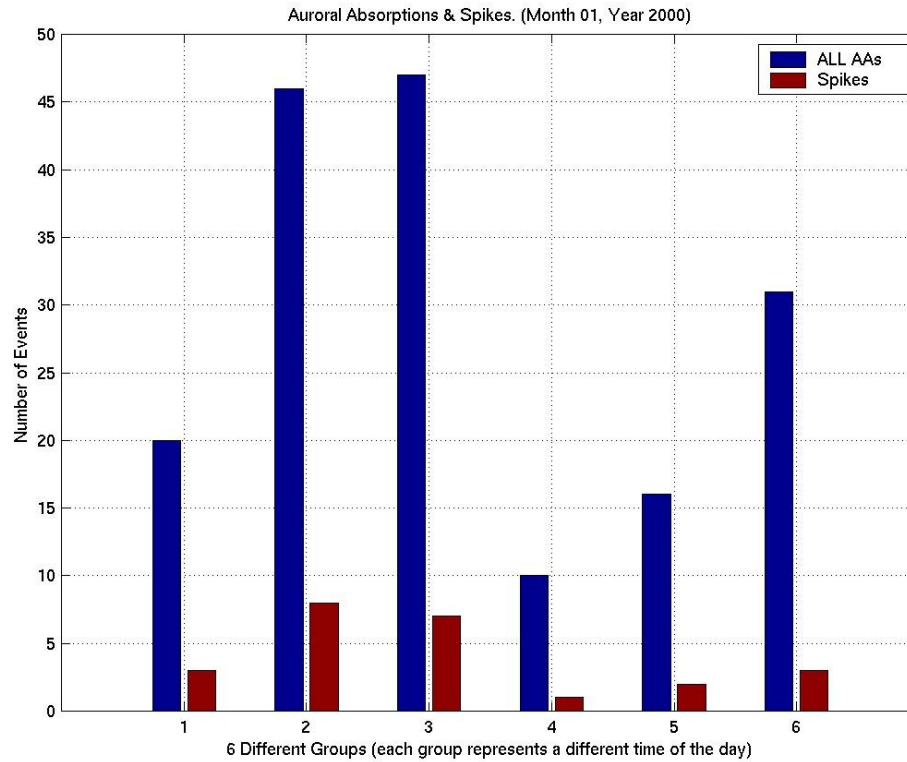


Figure 5.1.64: Statistical Analysis I, Month 01 Year 2000.

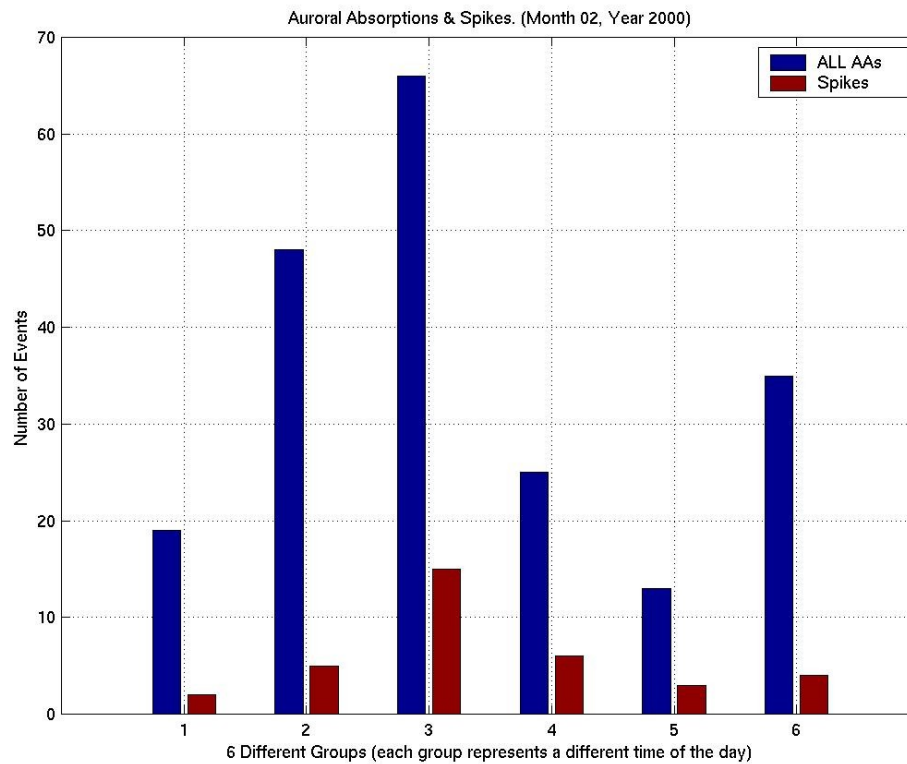


Figure 5.1.65: Statistical Analysis I, Month 02 Year 2000.

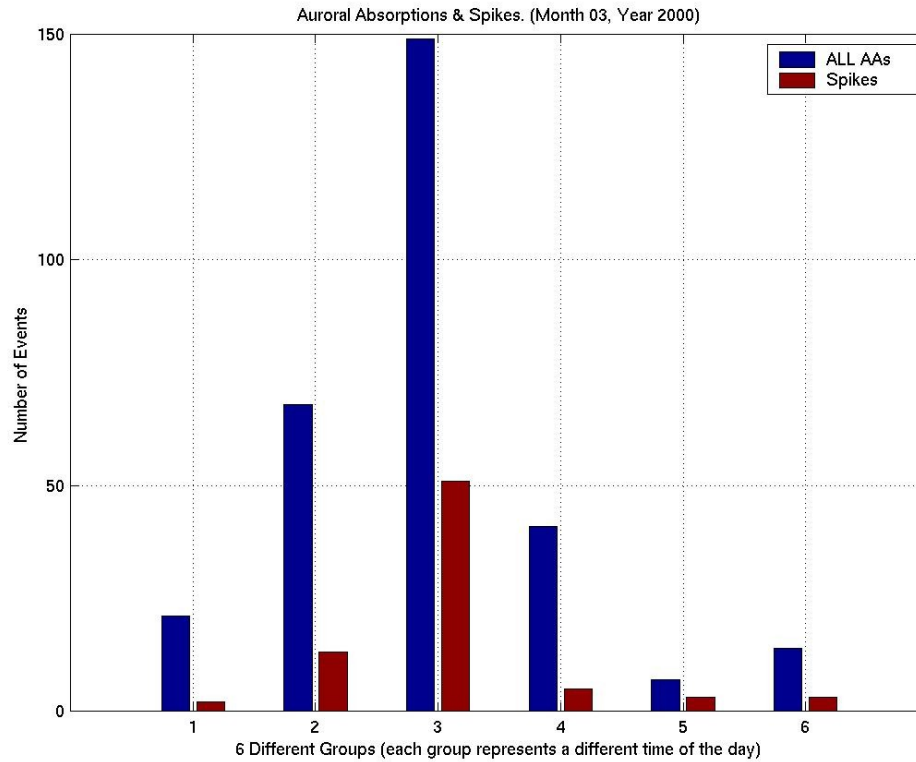


Figure 5.1.66: Statistical Analysis I, Month 03 Year 2000.

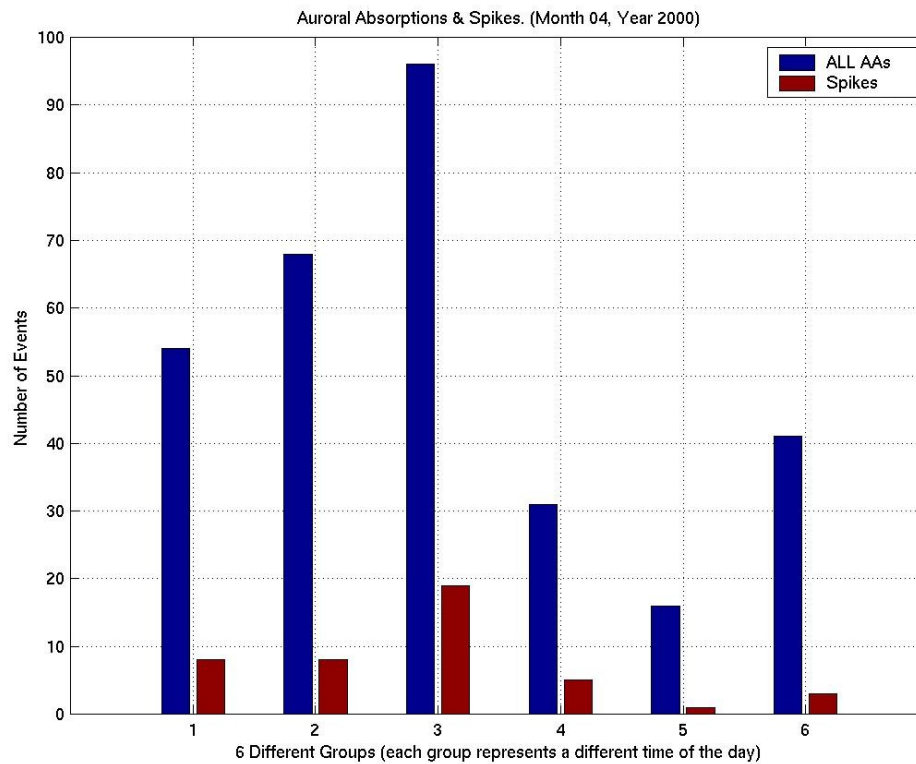


Figure 5.1.67: Statistical Analysis I, Month 04 Year 2000.

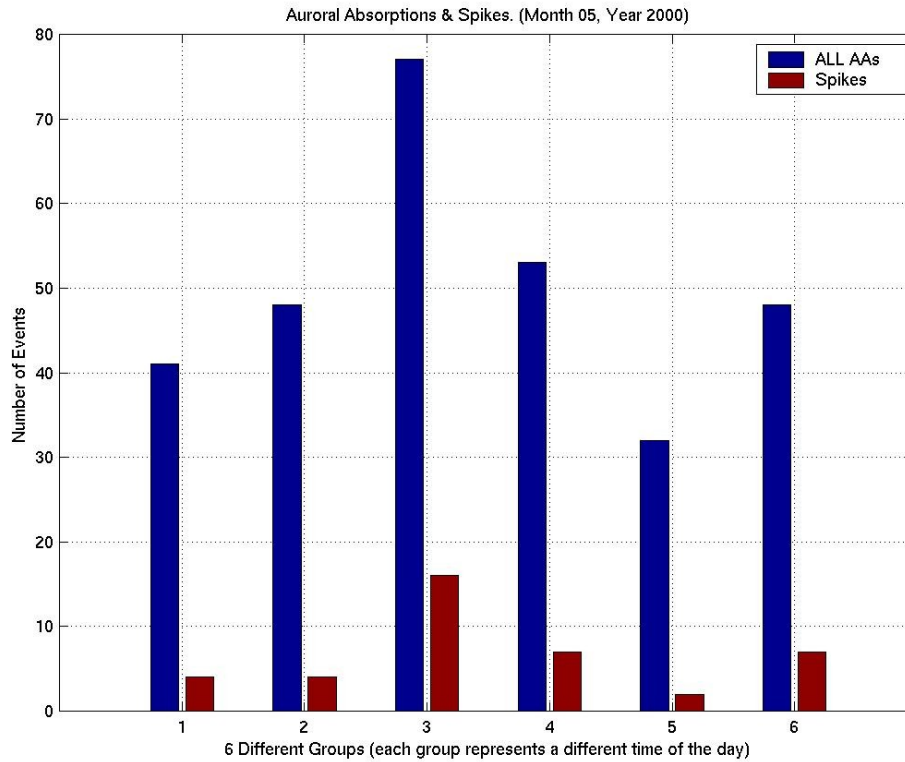


Figure 5.1.68: Statistical Analysis I, Month 05 Year 2000.

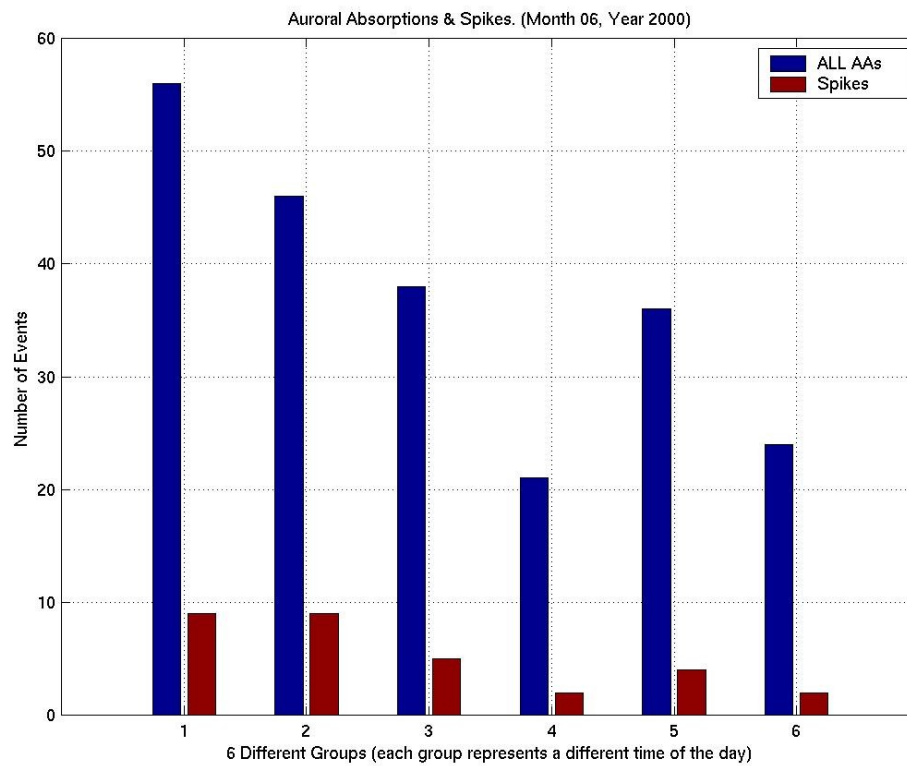


Figure 5.1.69: Statistical Analysis I, Month 06 Year 2000.

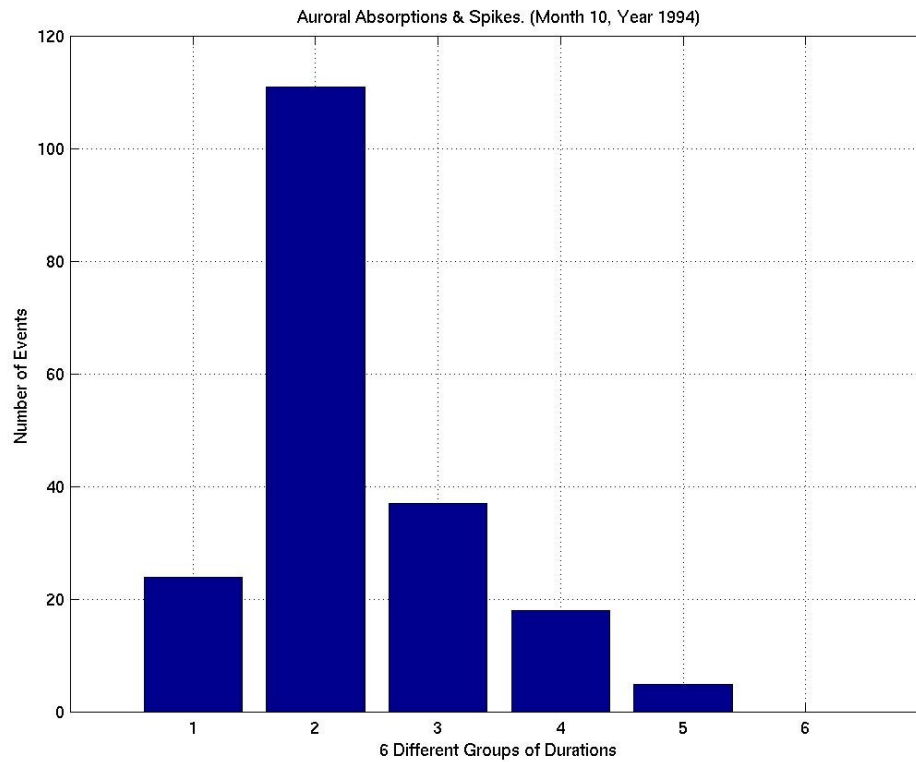


Figure 5.2.1: Statistical Analysis II, Month 10 Year 1994.

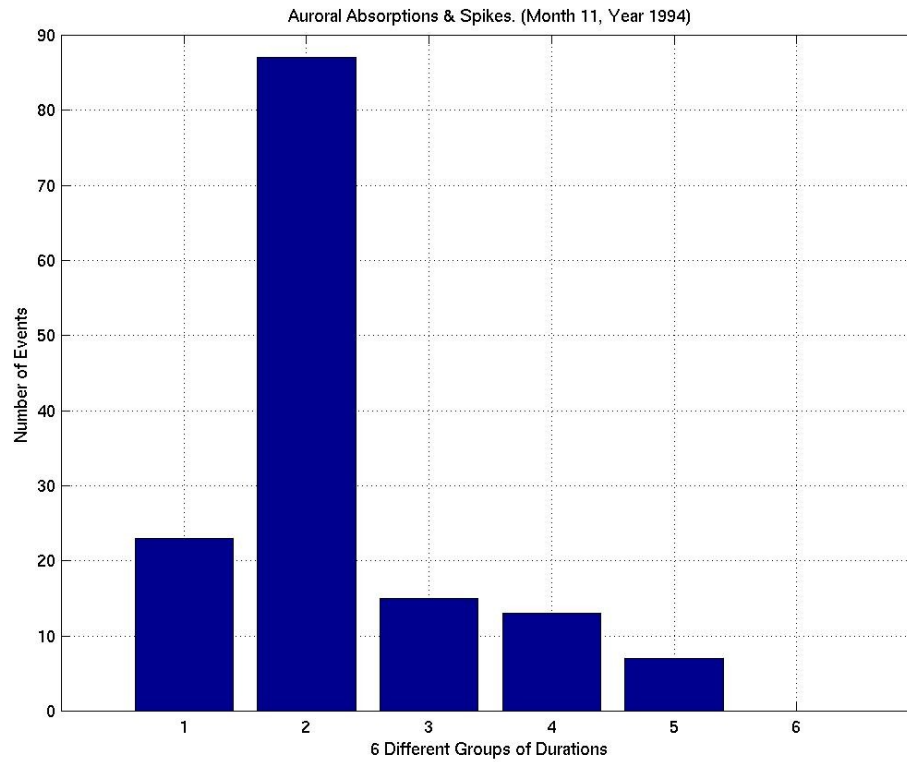


Figure 5.2.2: Statistical Analysis II, Month 11 Year 1994.

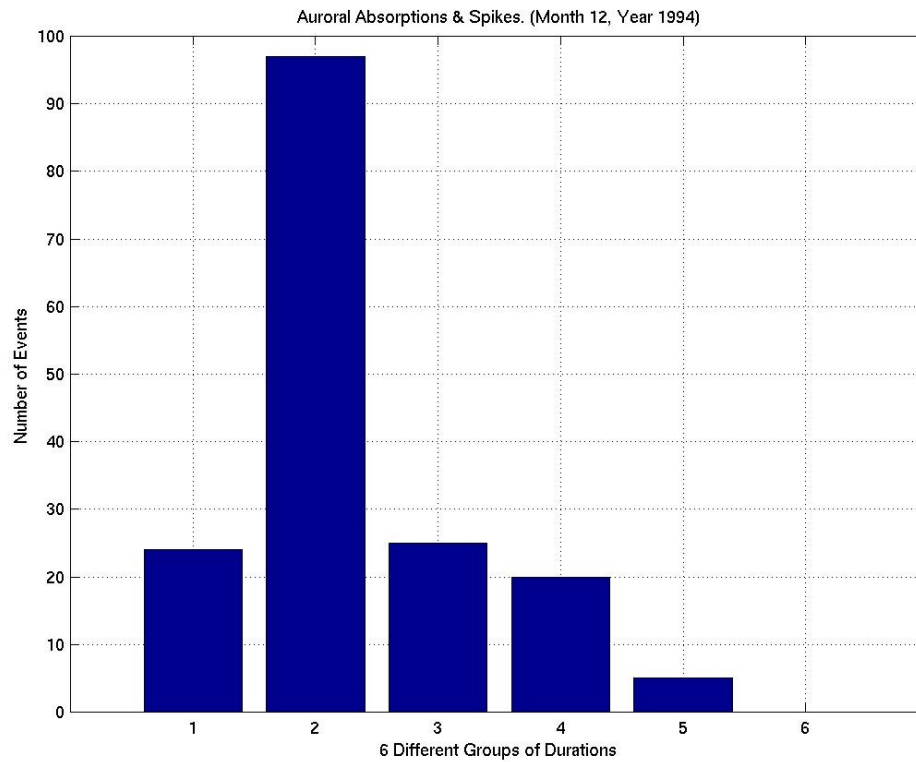


Figure 5.2.3: Statistical Analysis II, Month 12 Year 1994.

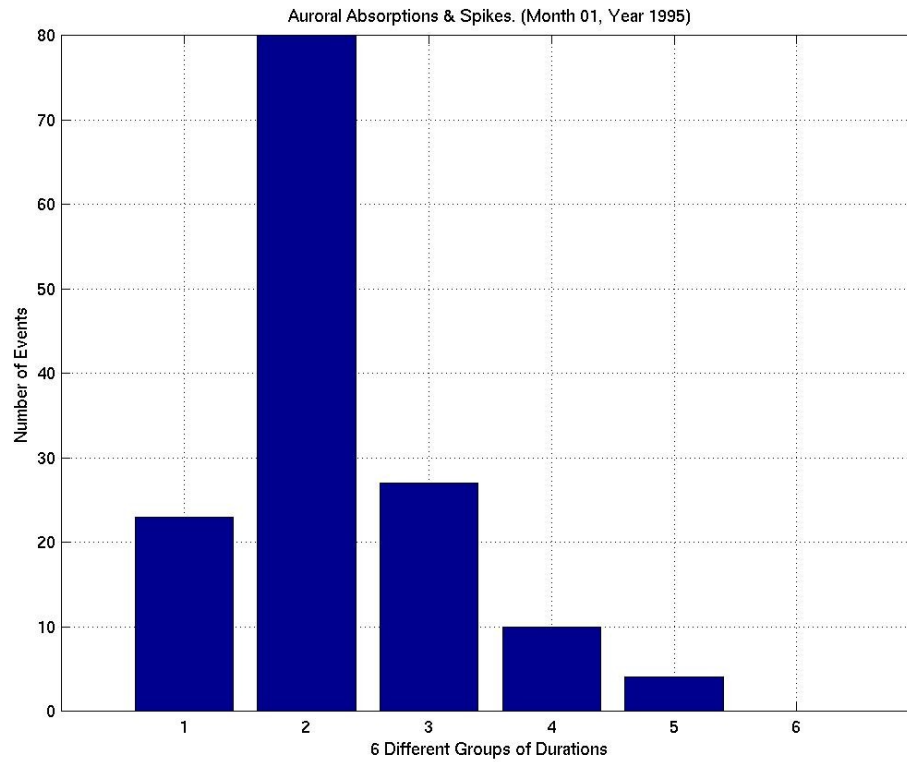


Figure 5.2.4: Statistical Analysis II, Month 01 Year 1995.

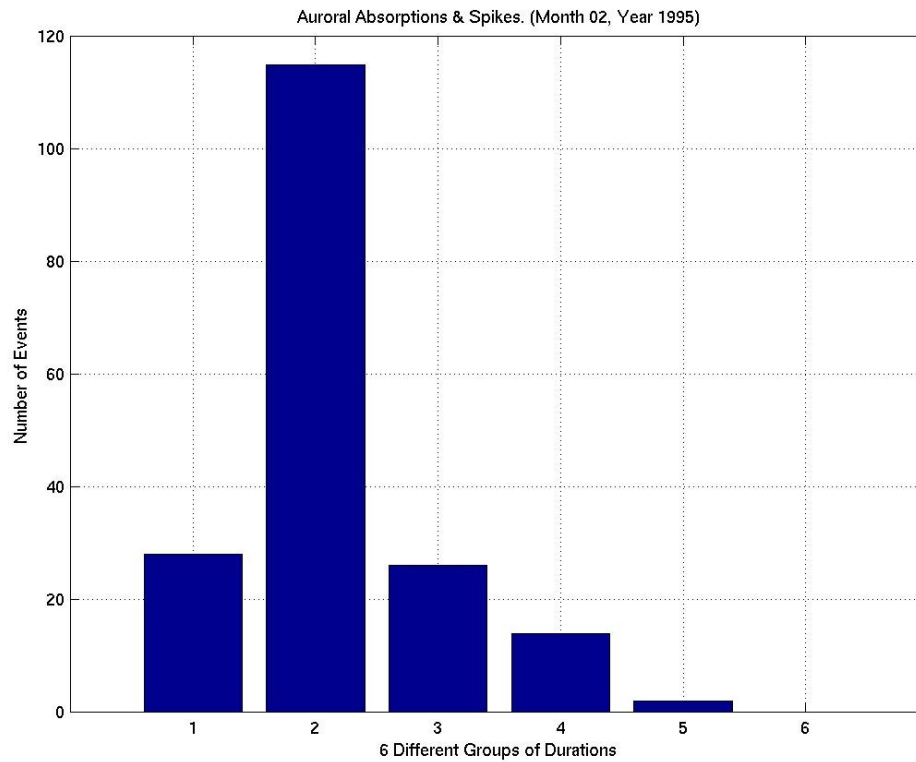


Figure 5.2.5: Statistical Analysis II, Month 02 Year 1995.

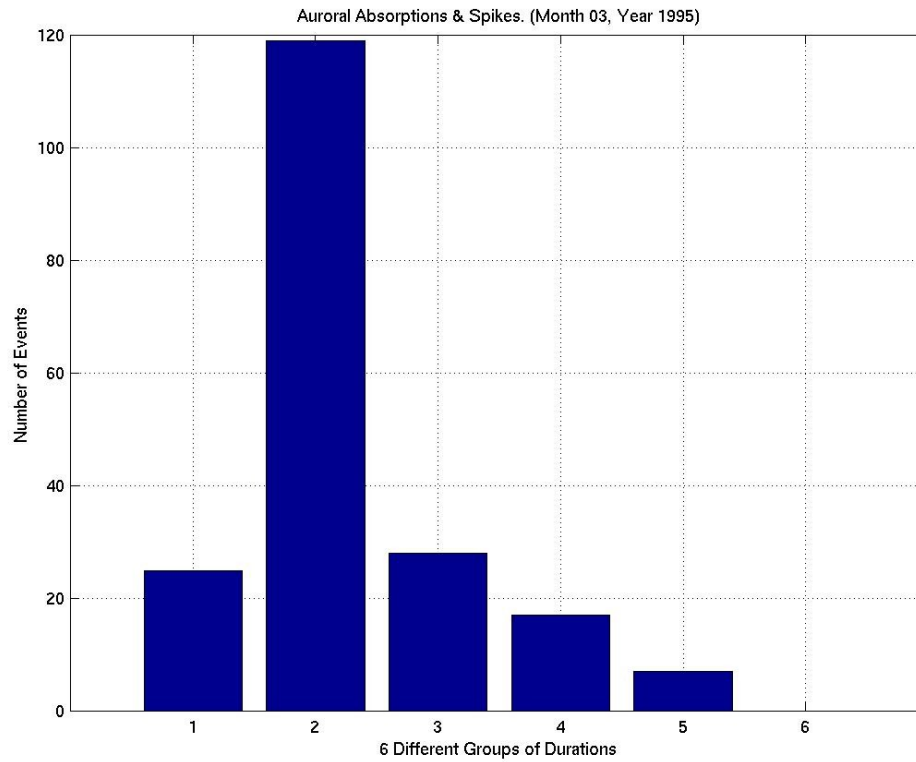


Figure 5.2.6: Statistical Analysis II, Month 03 Year 1995.

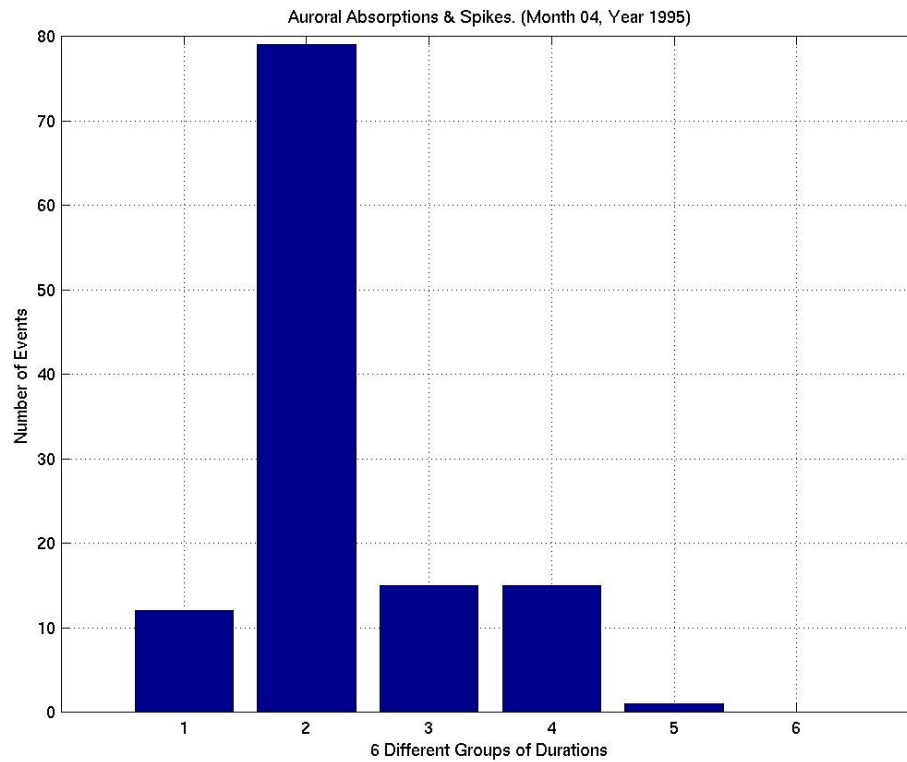


Figure 5.2.7: Statistical Analysis II, Month 04 Year 1995.

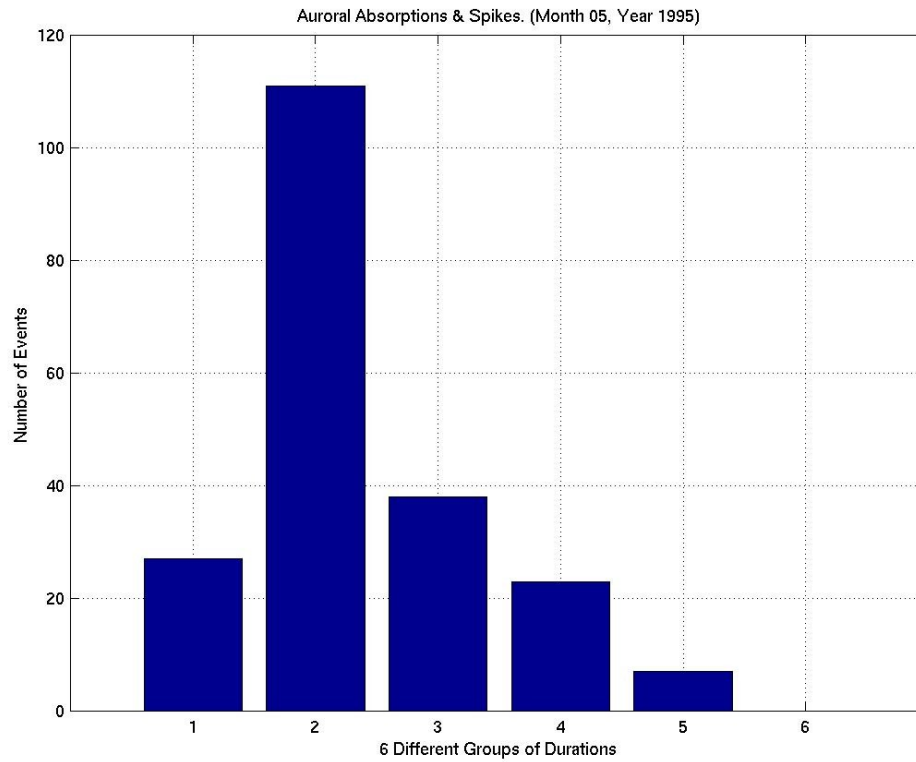


Figure 5.2.8: Statistical Analysis II, Month 05 Year 1995.

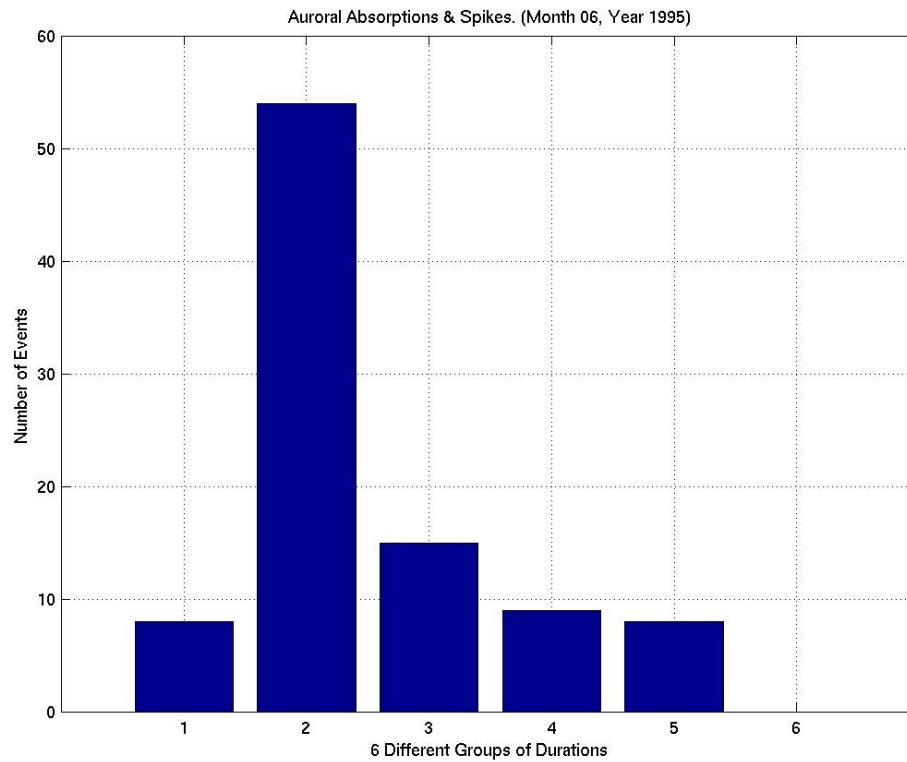


Figure 5.2.9: Statistical Analysis II, Month 06 Year 1995.

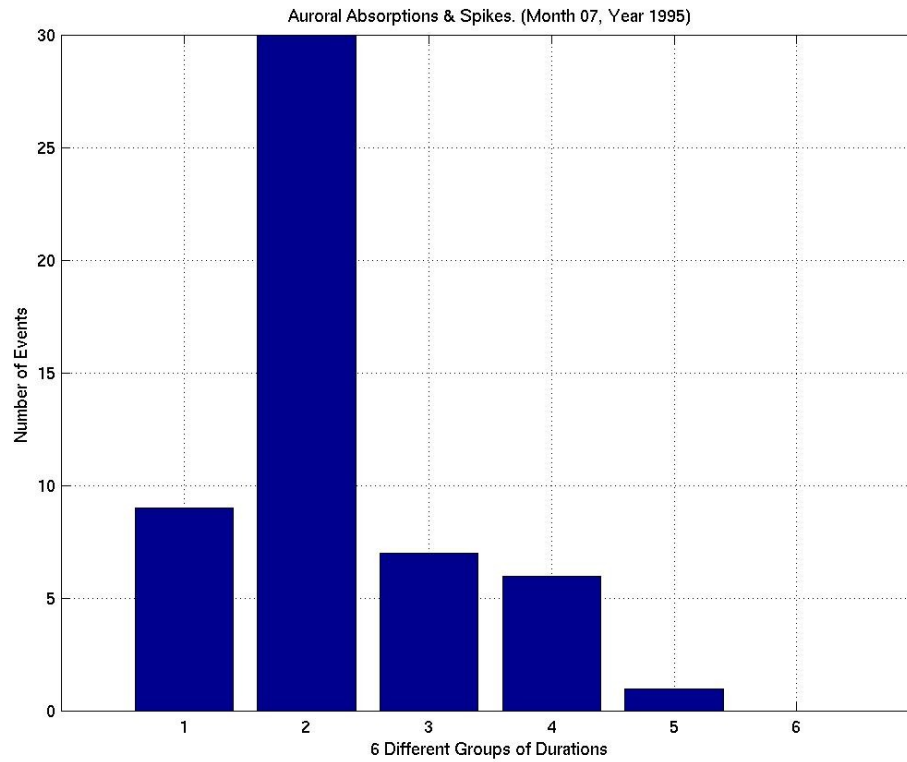


Figure 5.2.10: Statistical Analysis II, Month 07 Year 1995.

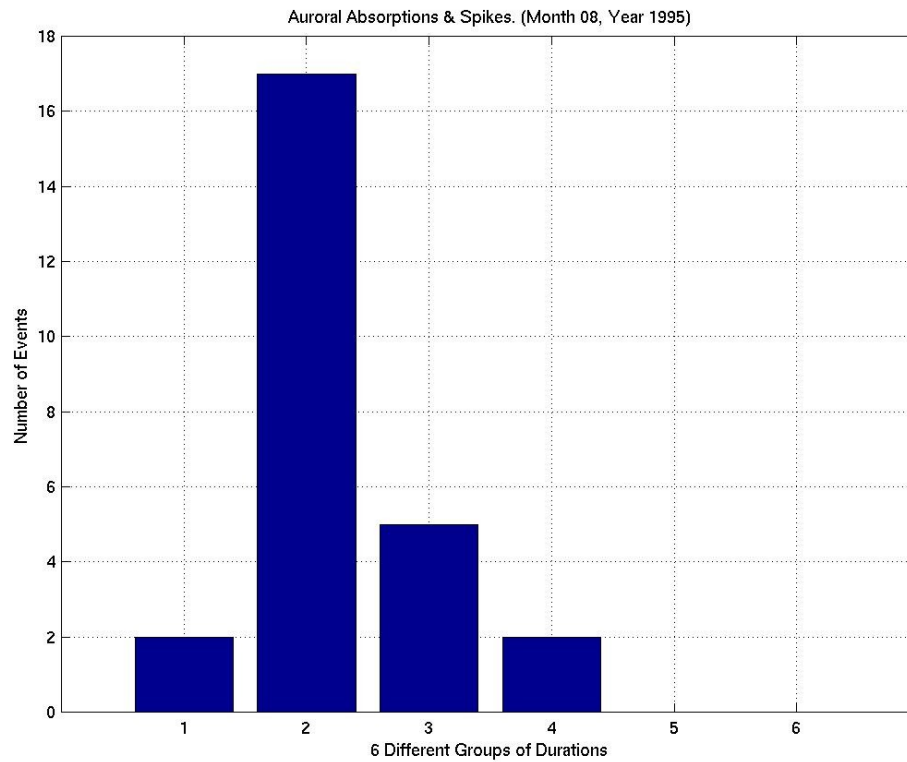


Figure 5.2.11: Statistical Analysis II, Month 08 Year 1995.

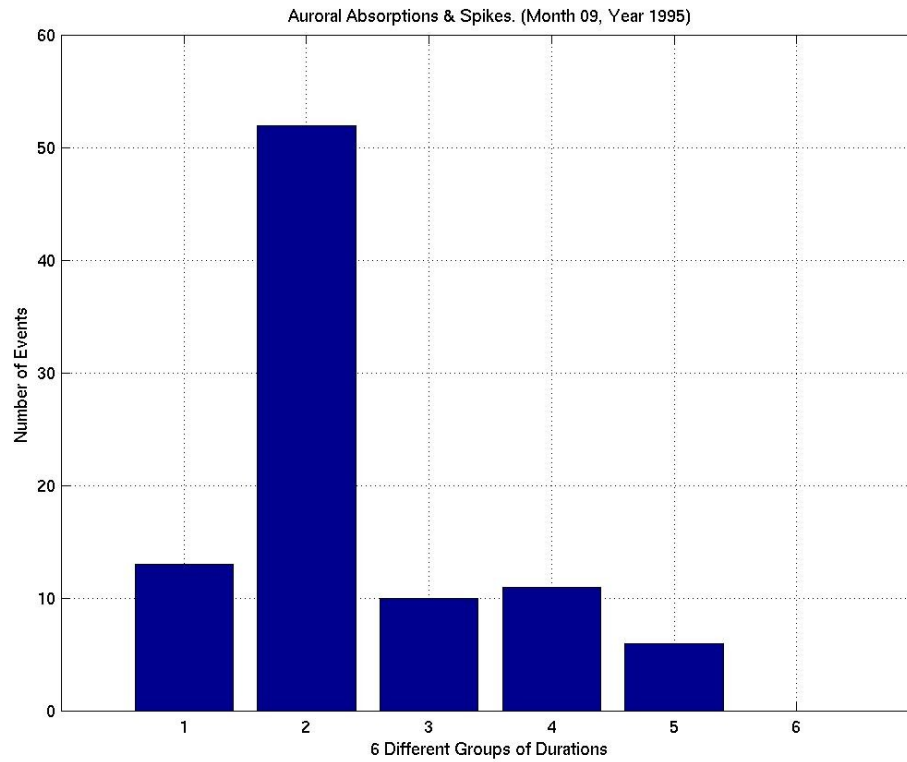


Figure 5.2.12: Statistical Analysis II, Month 09 Year 1995.

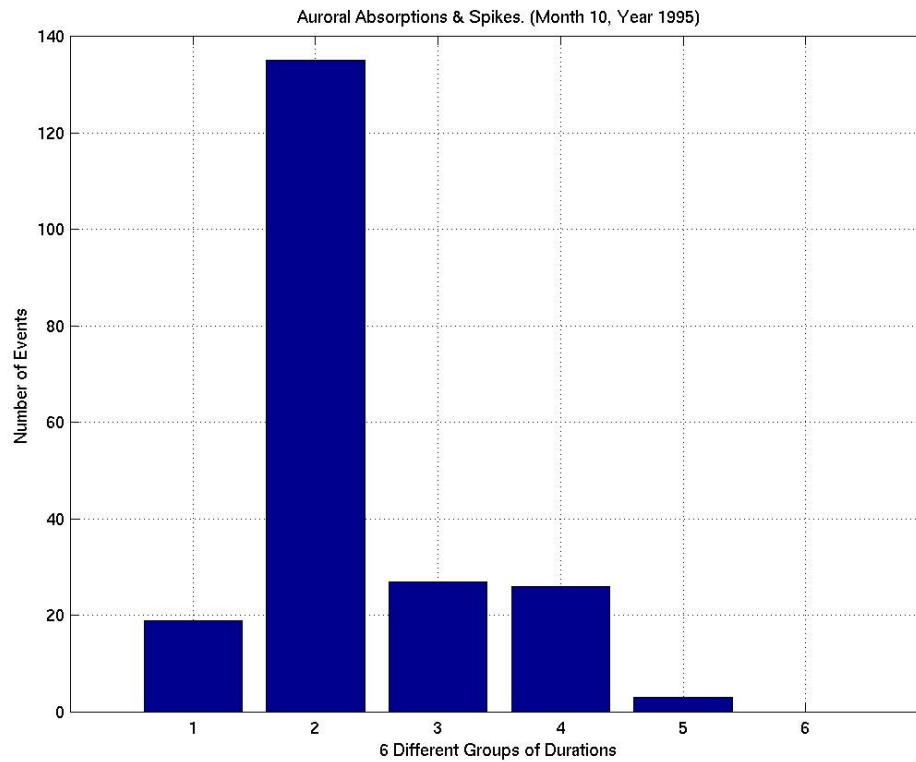


Figure 5.2.13: Statistical Analysis II, Month 10 Year 1995.

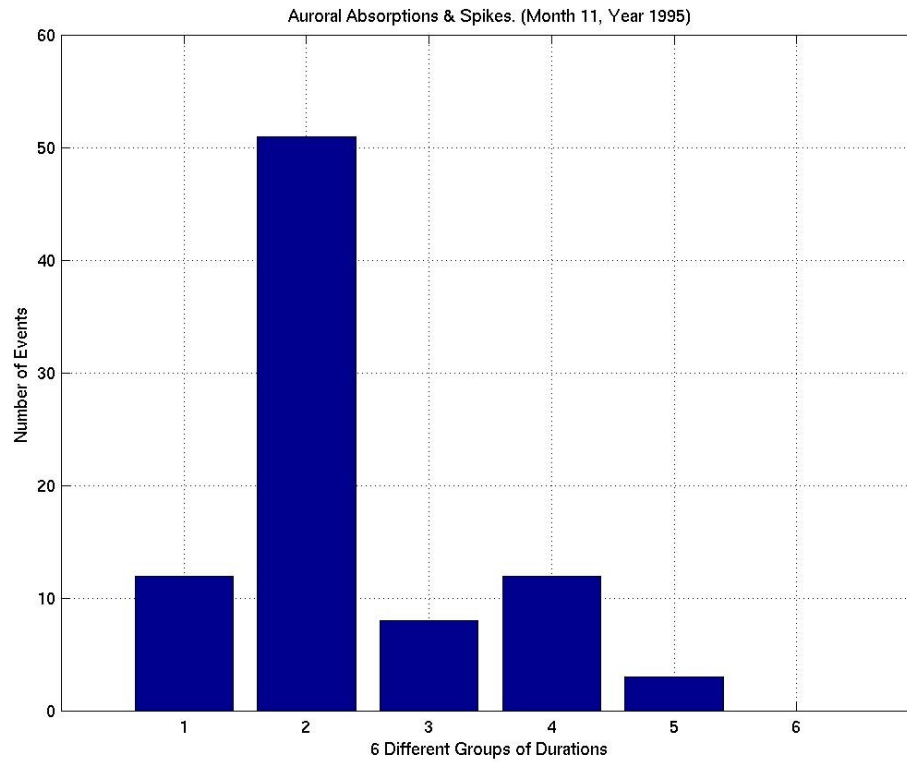


Figure 5.2.14: Statistical Analysis II, Month 11 Year 1995.

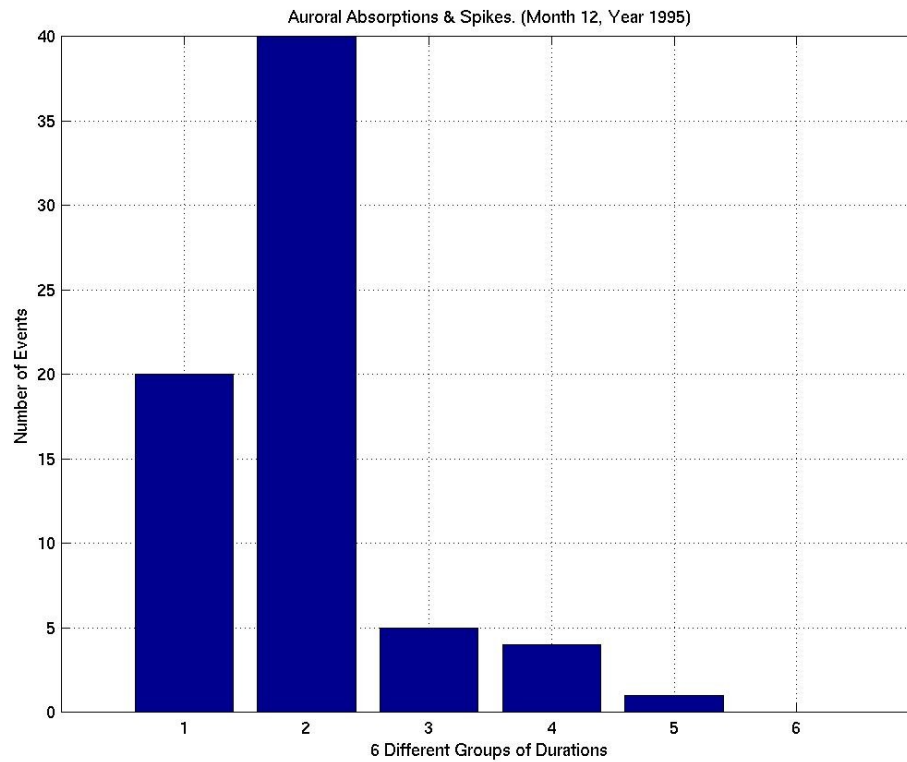


Figure 5.2.15: Statistical Analysis II, Month 12 Year 1995.

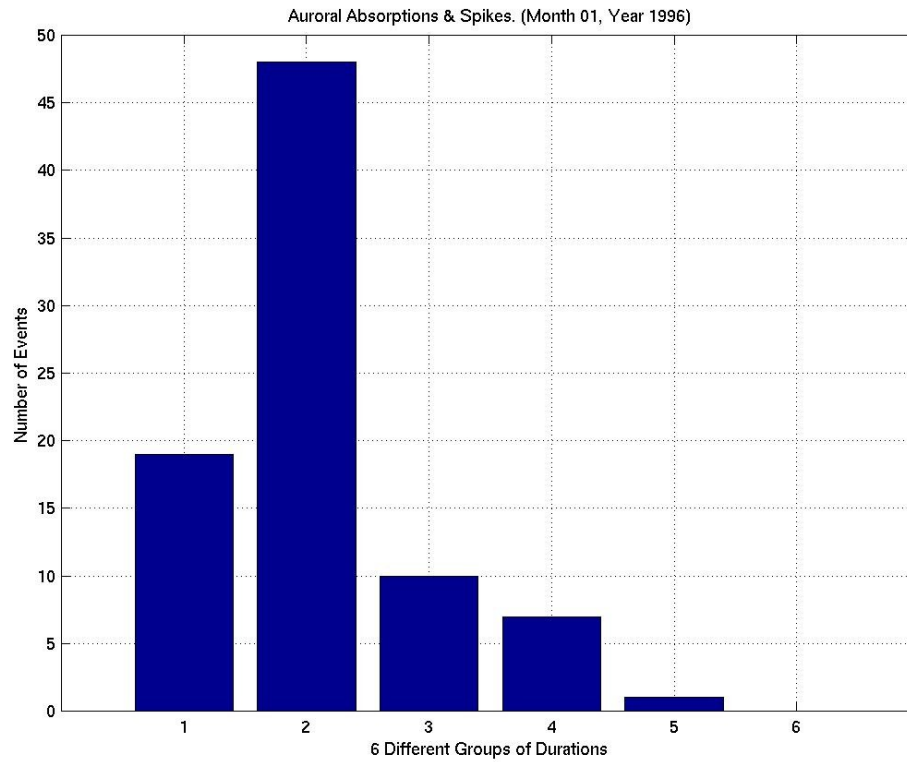


Figure 5.2.16: Statistical Analysis II, Month 01 Year 1996.

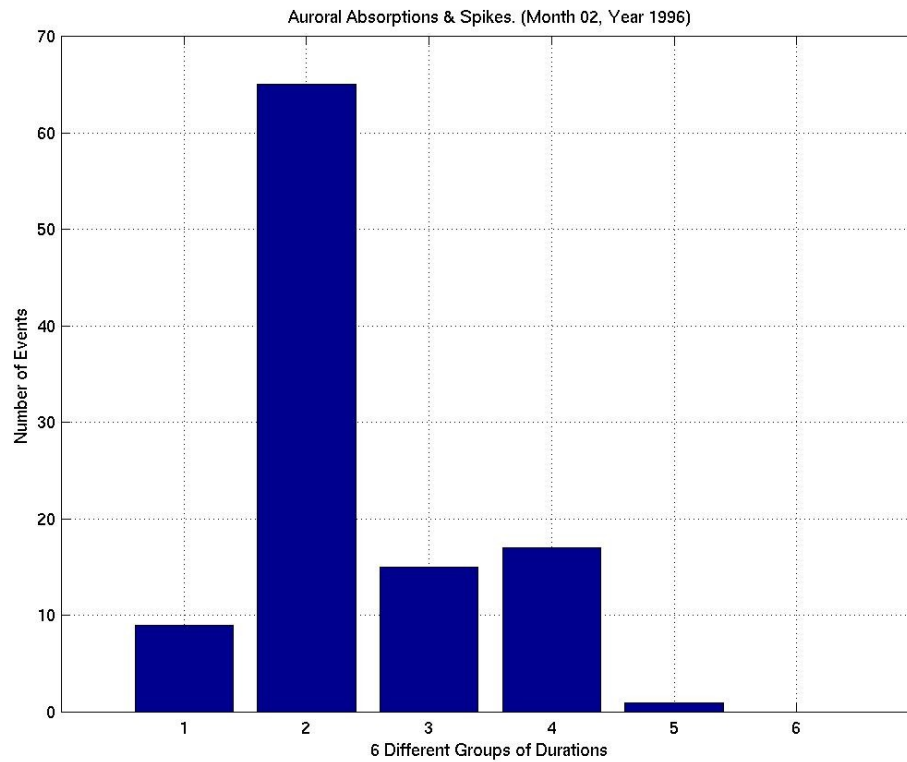


Figure 5.2.17: Statistical Analysis II, Month 02 Year 1996.

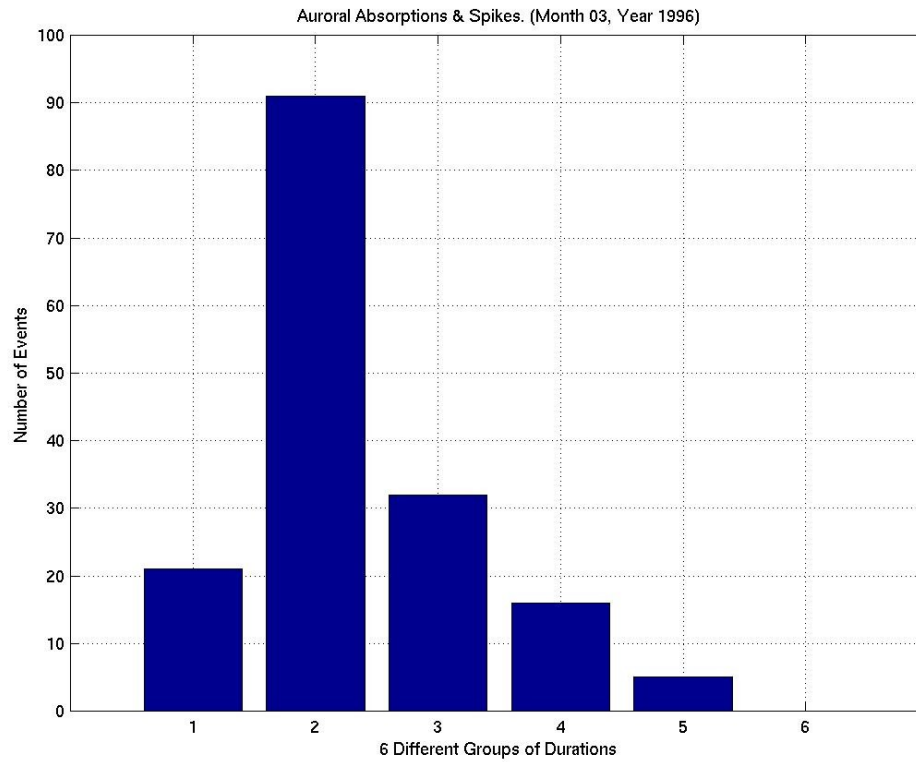


Figure 5.2.18: Statistical Analysis II, Month 03 Year 1996.

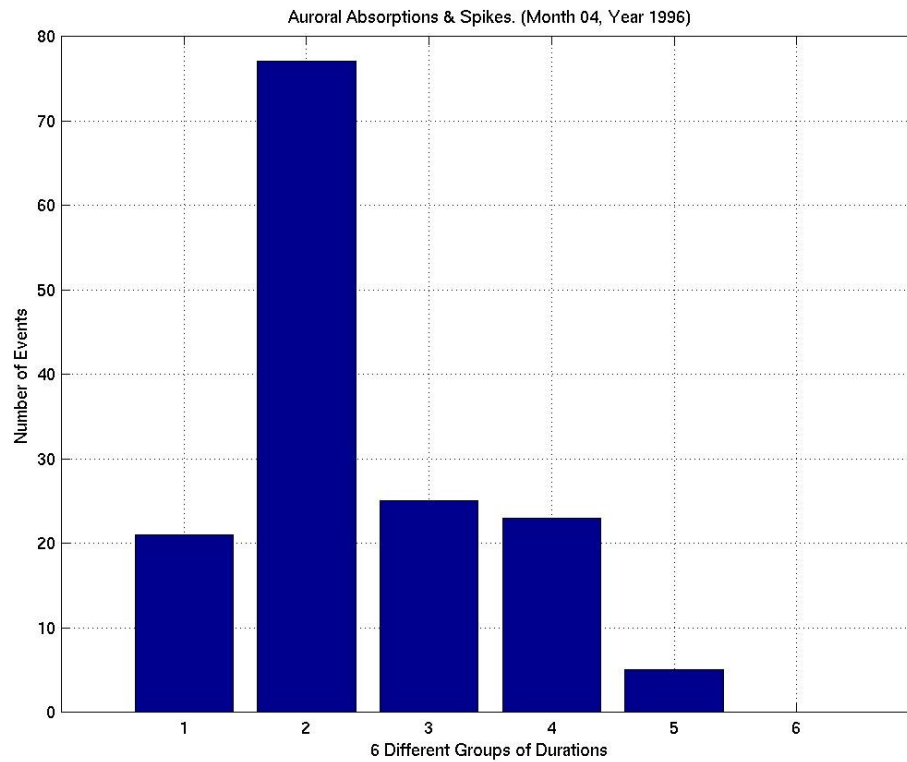


Figure 5.2.19: Statistical Analysis II, Month 04 Year 1996.

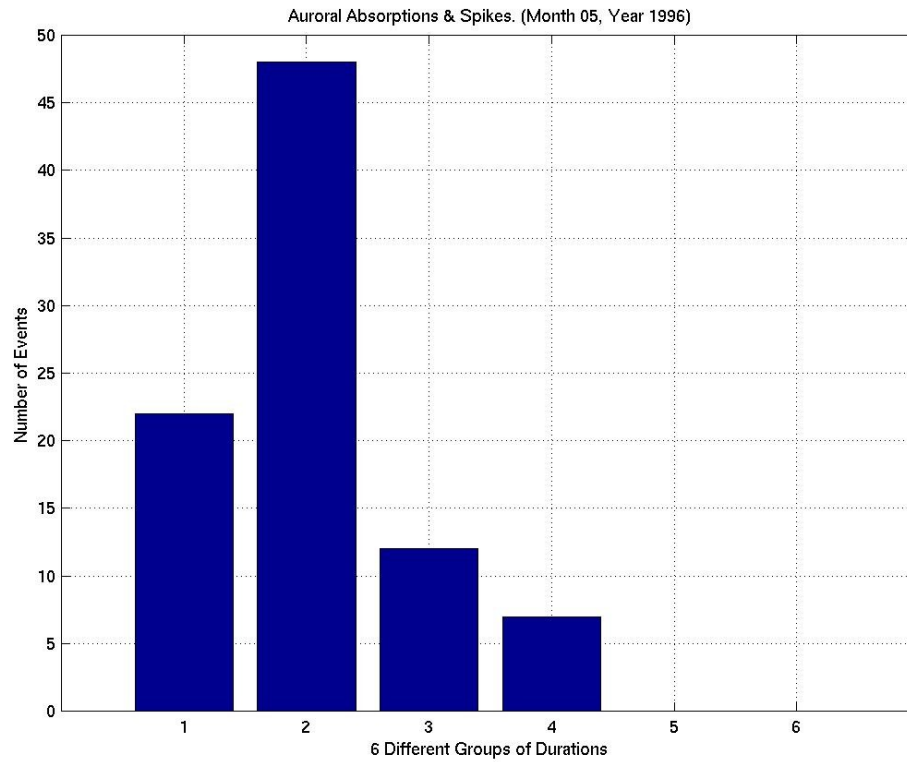


Figure 5.2.20: Statistical Analysis II, Month 05 Year 1996.

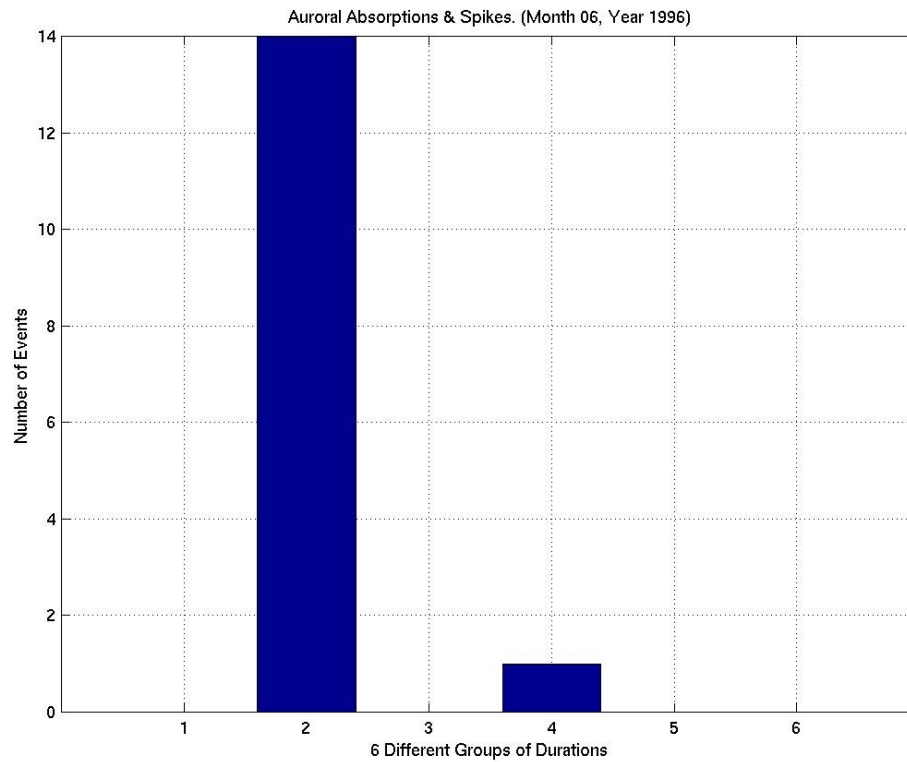


Figure 5.2.21: Statistical Analysis II, Month 06 Year 1996.

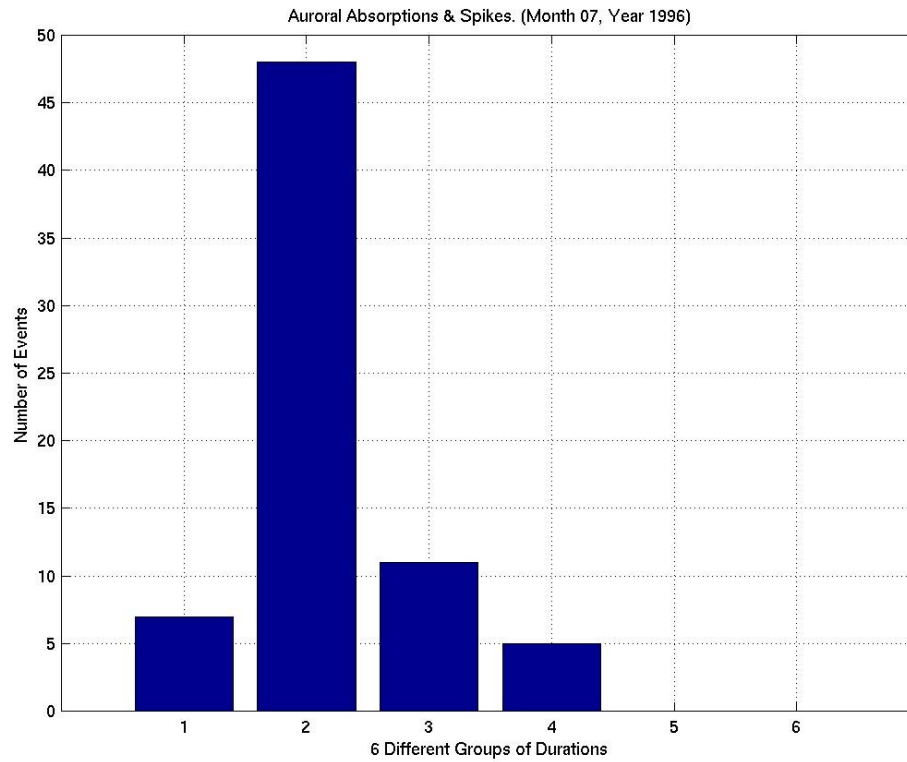


Figure 5.2.22: Statistical Analysis II, Month 07 Year 1996.

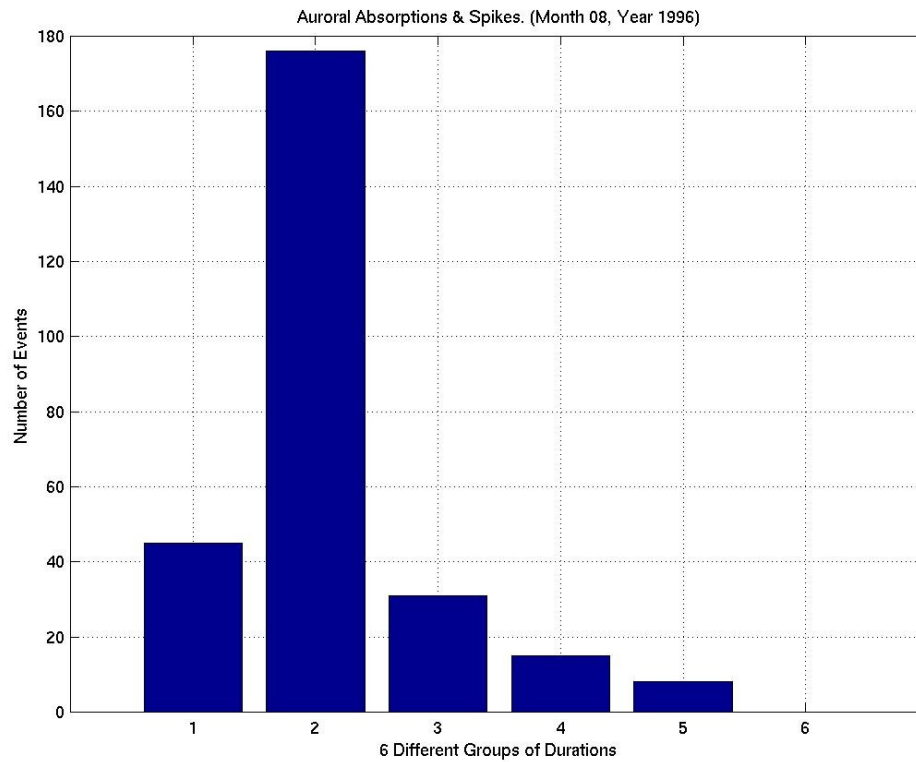


Figure 5.2.23: Statistical Analysis II, Month 08 Year 1996.

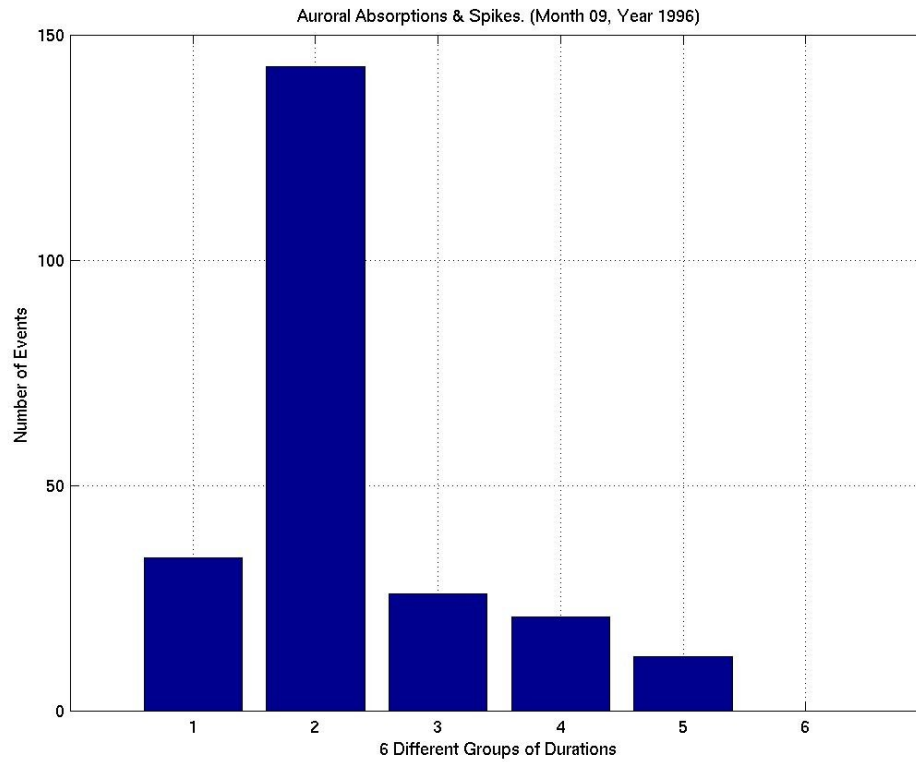


Figure 5.2.24: Statistical Analysis II, Month 09 Year 1996.

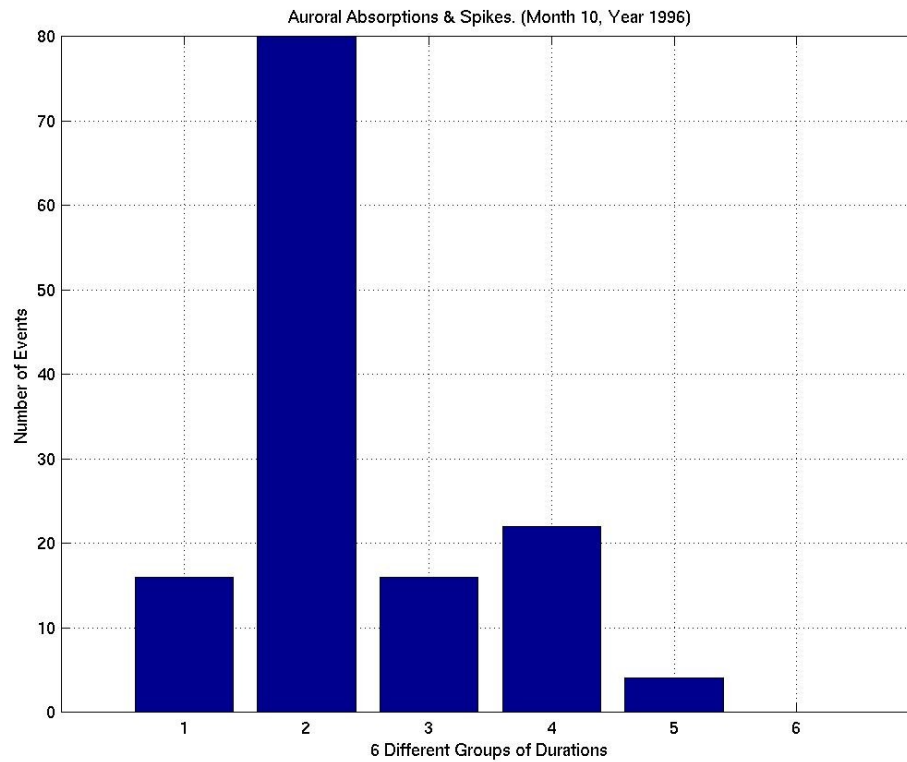


Figure 5.2.25: Statistical Analysis II, Month 10 Year 1996.

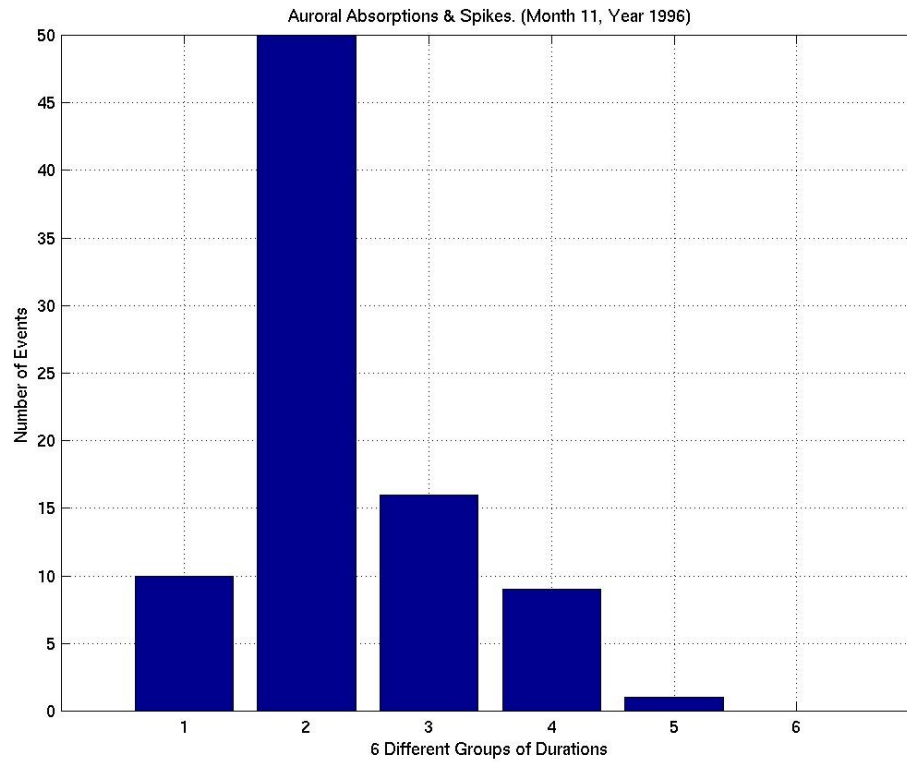


Figure 5.2.26: Statistical Analysis II, Month 11 Year 1996.

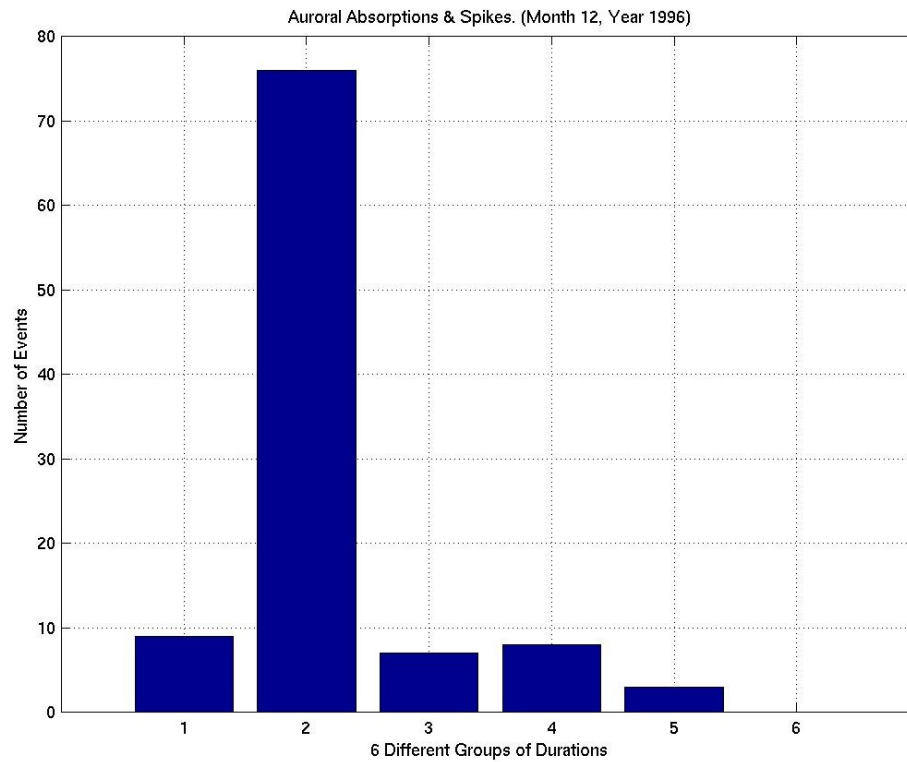


Figure 5.2.27: Statistical Analysis II, Month 12 Year 1996.

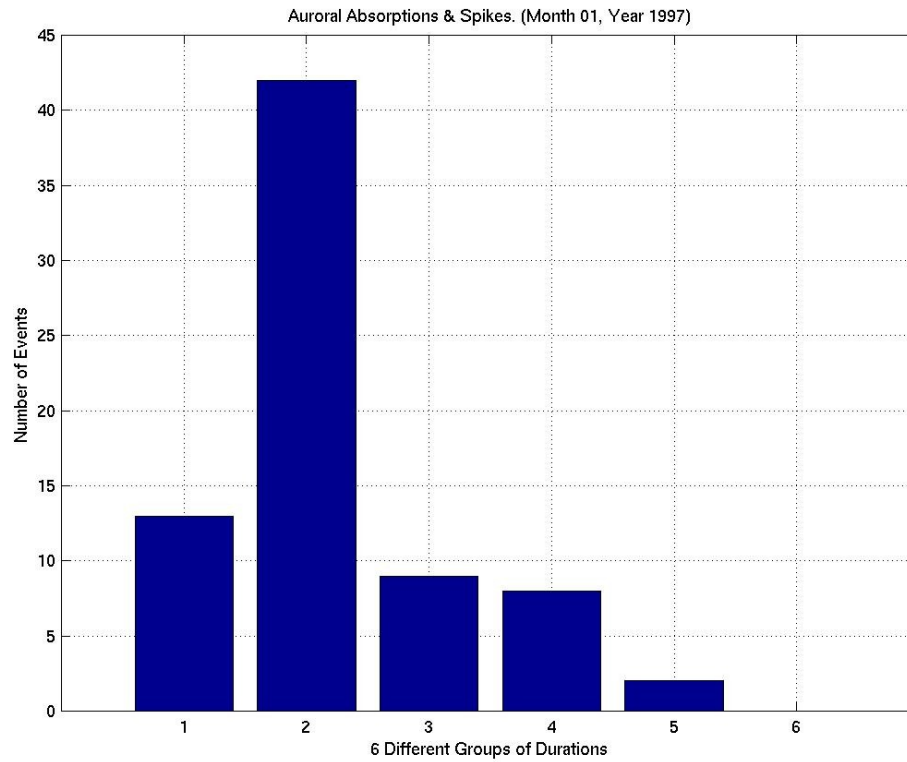


Figure 5.2.28: Statistical Analysis II, Month 01 Year 1997.

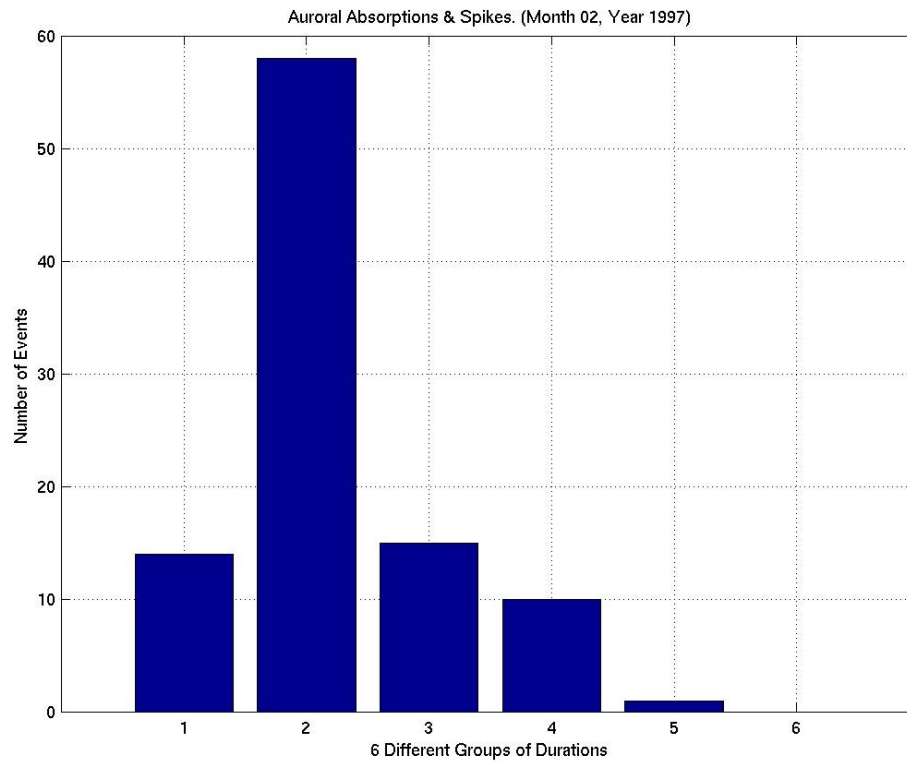


Figure 5.2.29: Statistical Analysis II, Month 02 Year 1997.

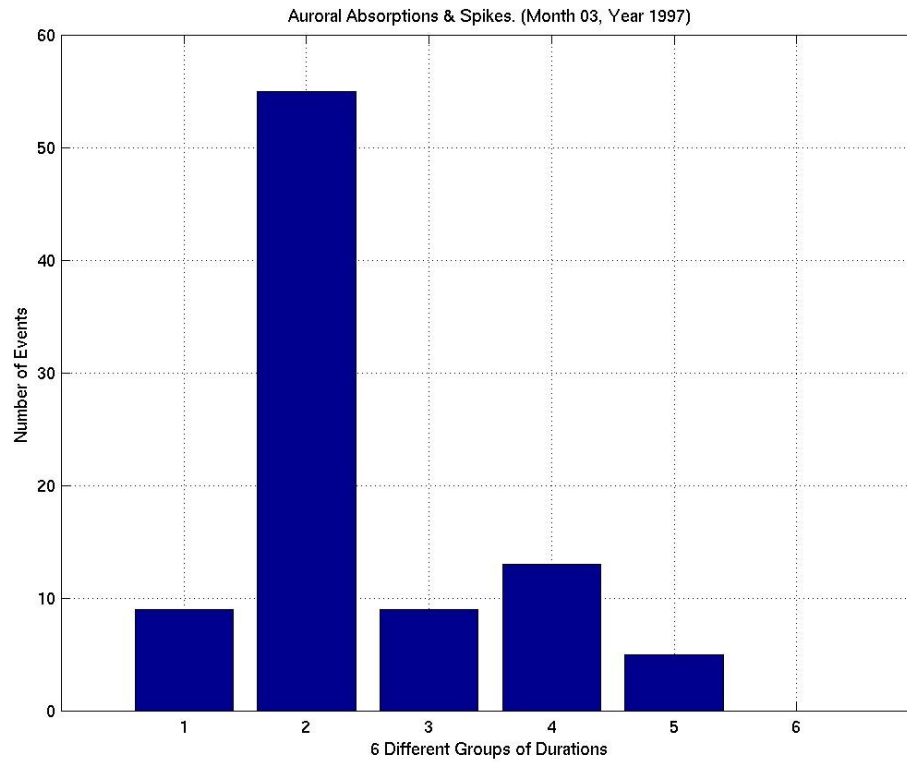


Figure 5.2.30: Statistical Analysis II, Month 03 Year 1997.

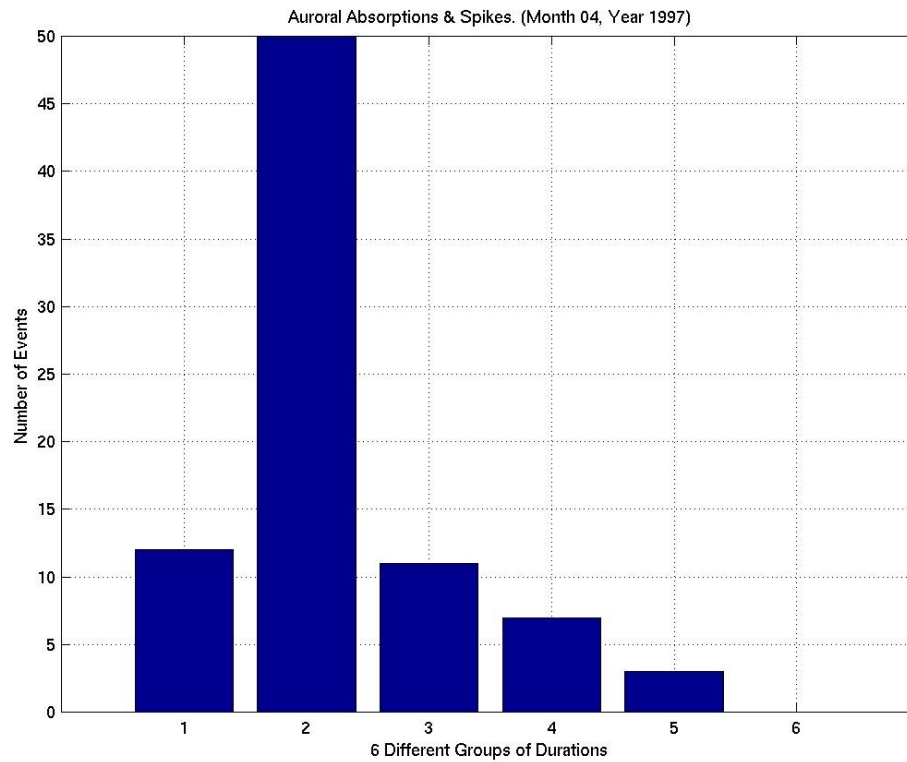


Figure 5.2.31: Statistical Analysis II, Month 04 Year 1997.

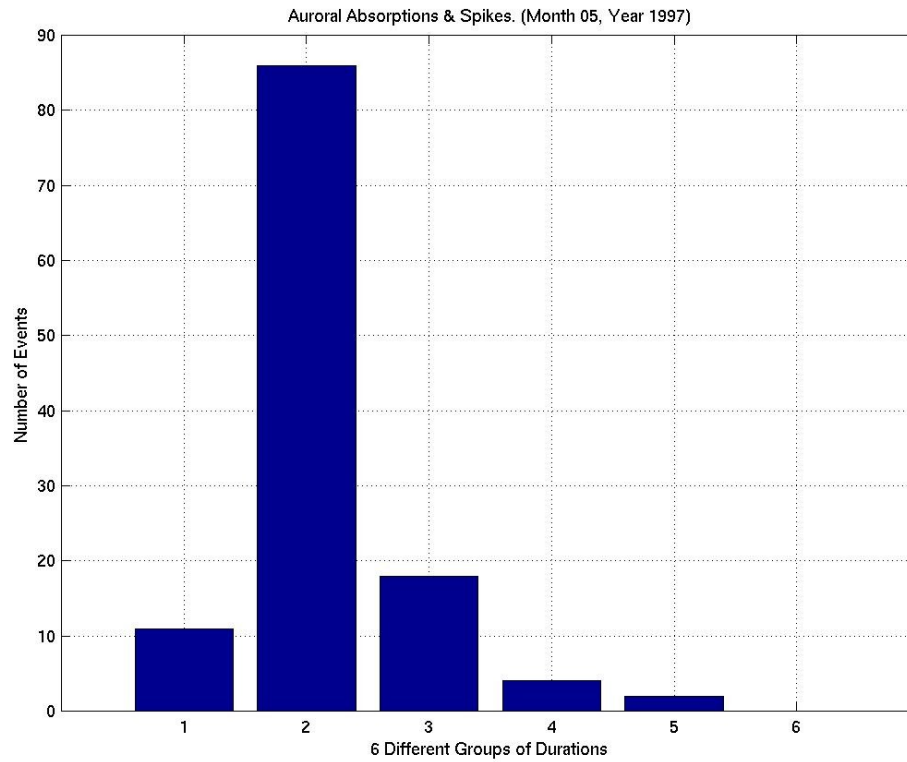


Figure 5.2.32: Statistical Analysis II, Month 05 Year 1997.

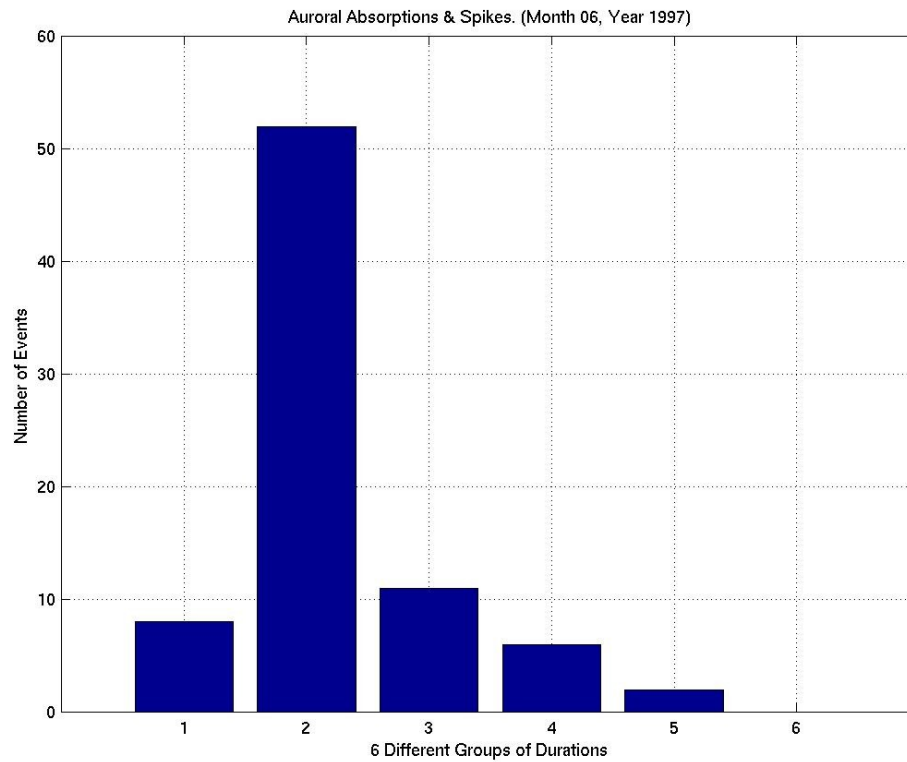


Figure 5.2.33: Statistical Analysis II, Month 06 Year 1997.

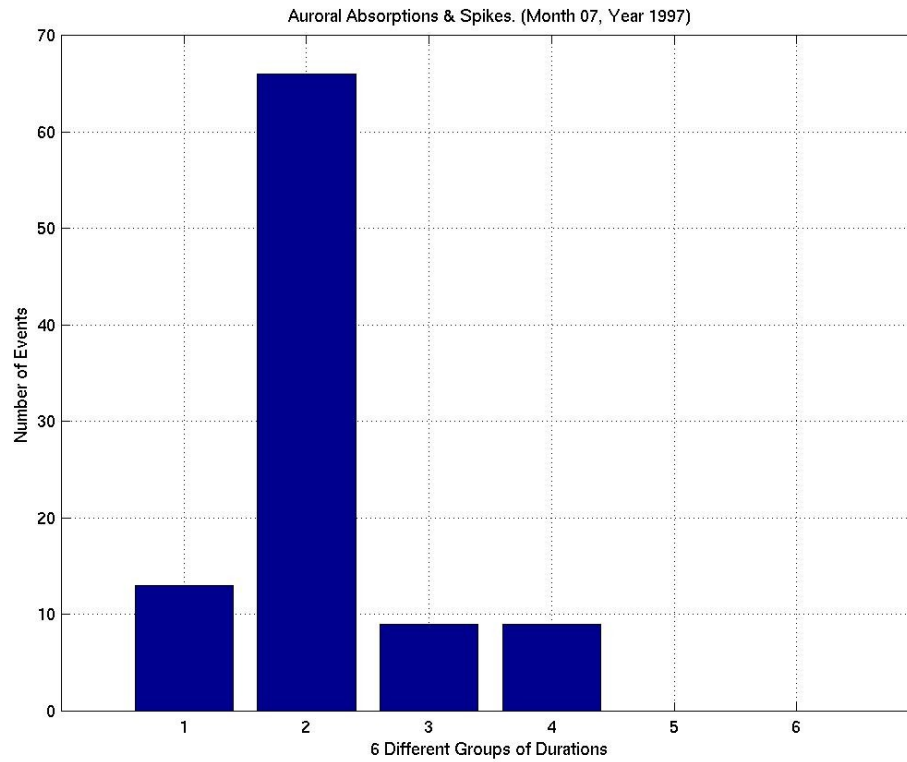


Figure 5.2.34: Statistical Analysis II, Month 07 Year 1997.

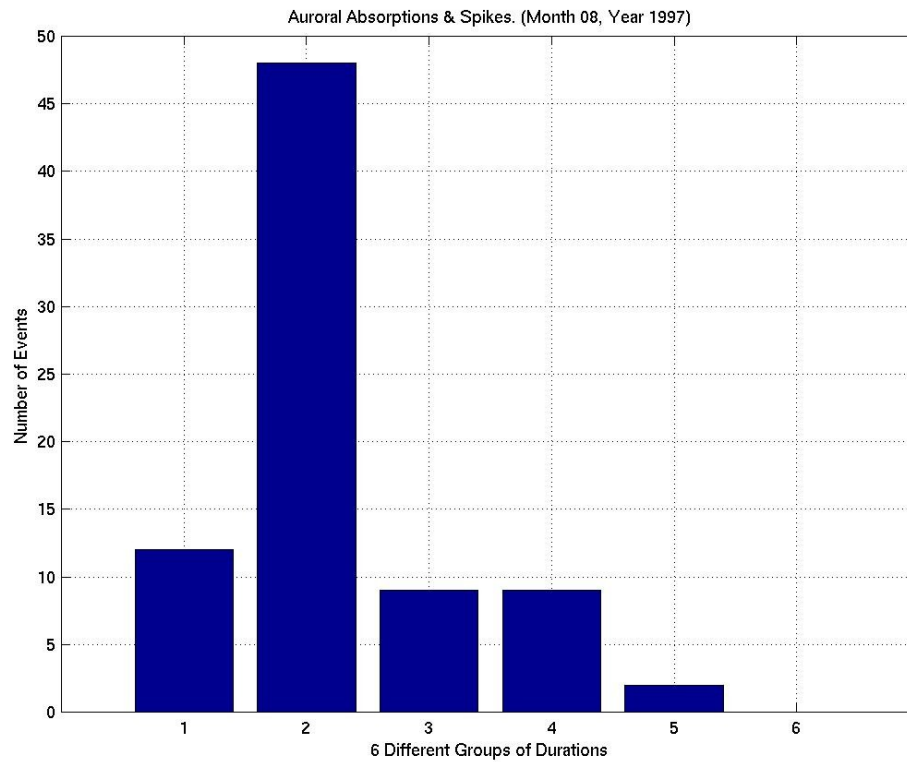


Figure 5.2.35: Statistical Analysis II, Month 08 Year 1997.

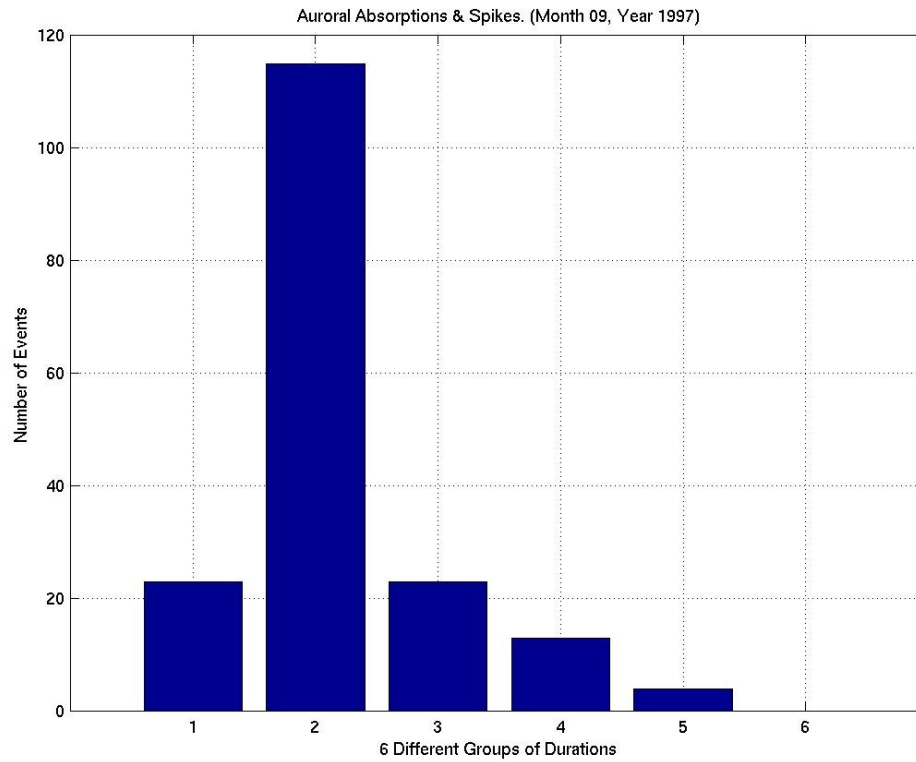


Figure 5.2.36: Statistical Analysis II, Month 09 Year 1997.

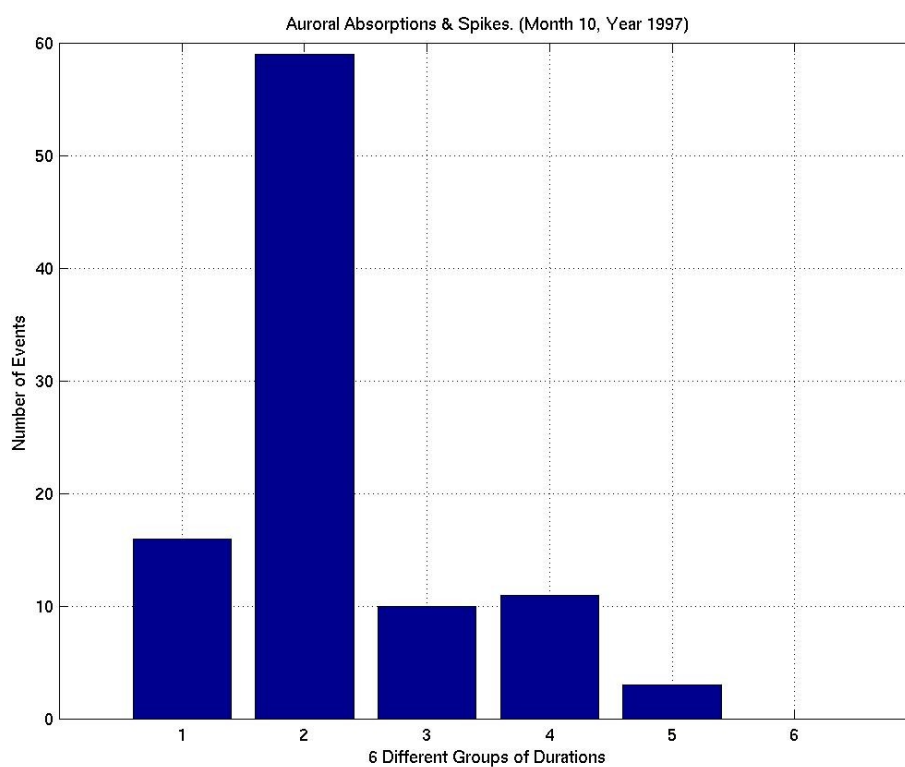


Figure 5.2.37: Statistical Analysis II, Month 10 Year 1997.

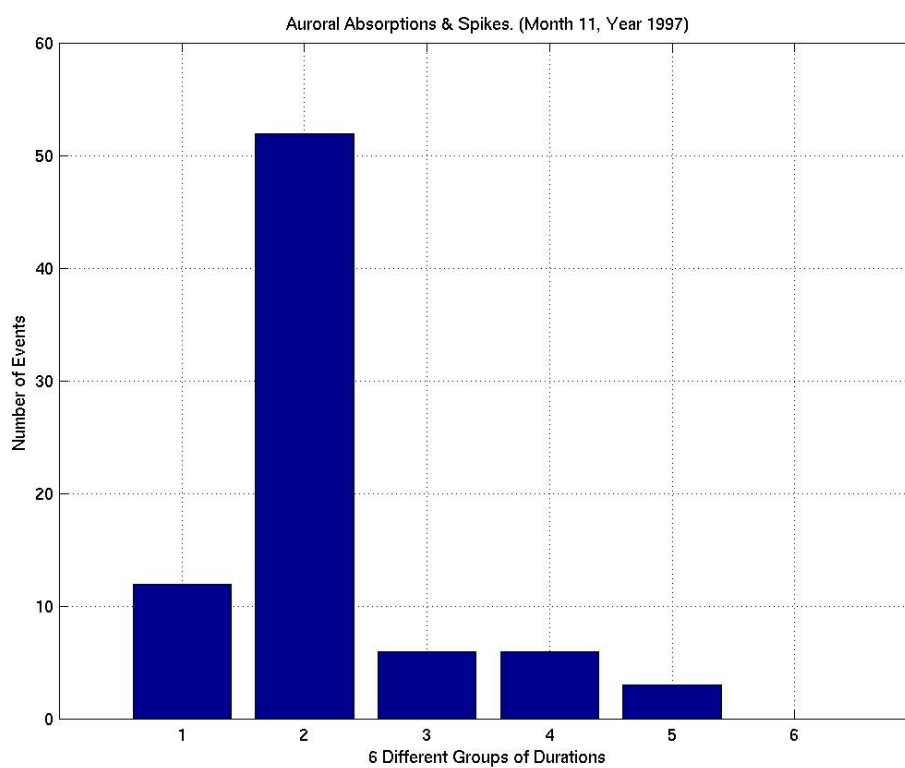


Figure 5.2.38: Statistical Analysis II, Month 11 Year 1997.

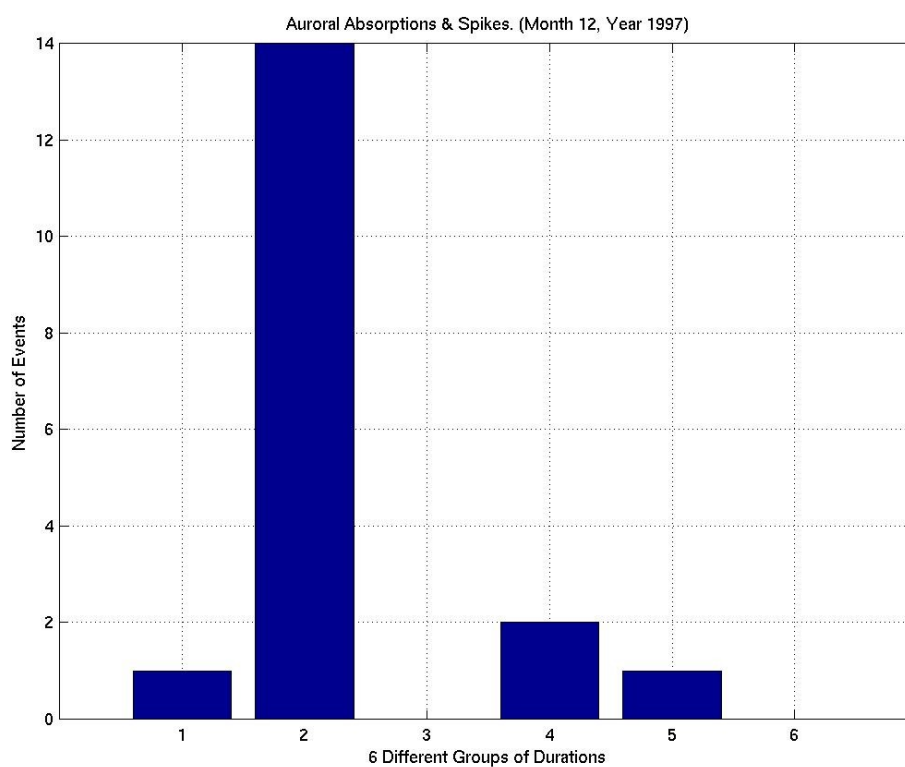


Figure 5.2.39: Statistical Analysis II, Month 12 Year 1997.

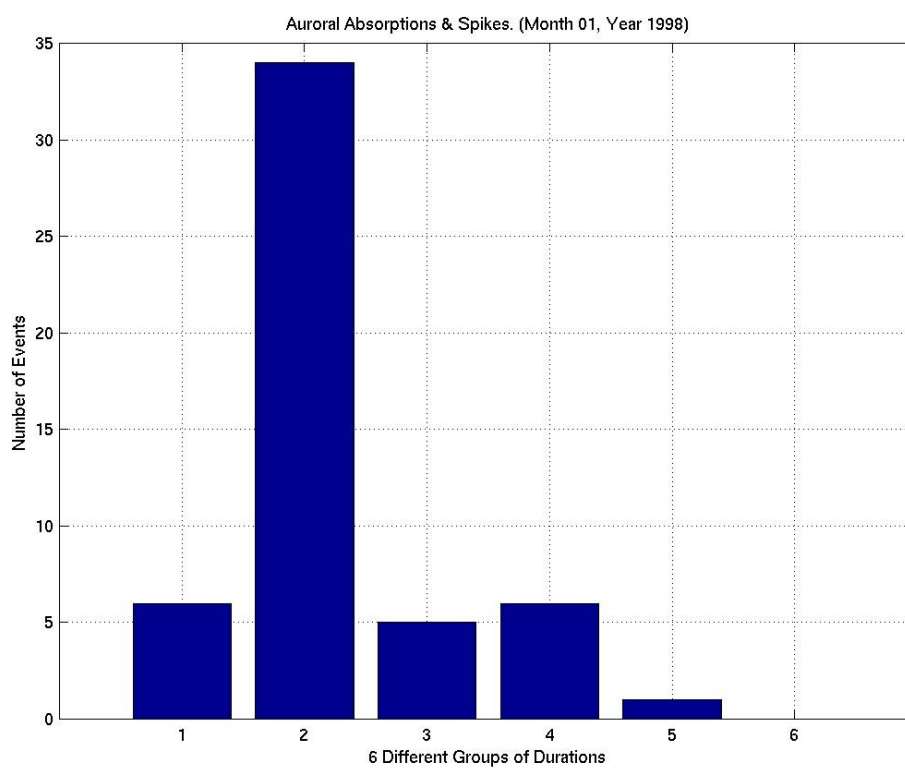


Figure 5.2.40: Statistical Analysis II, Month 01 Year 1998.

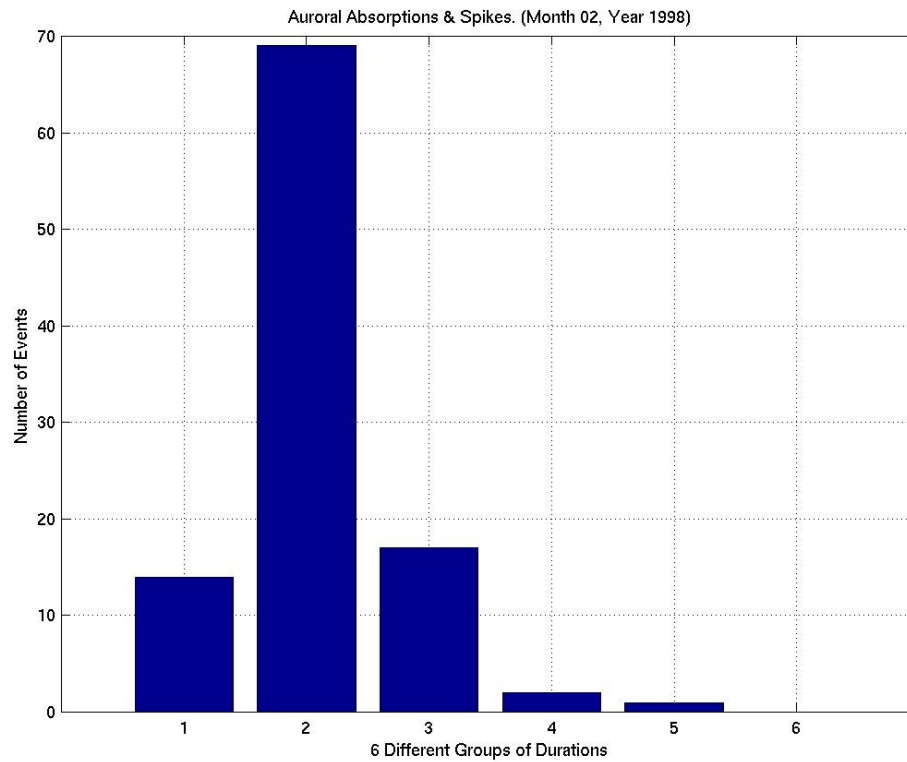


Figure 5.2.41: Statistical Analysis II, Month 02 Year 1998.

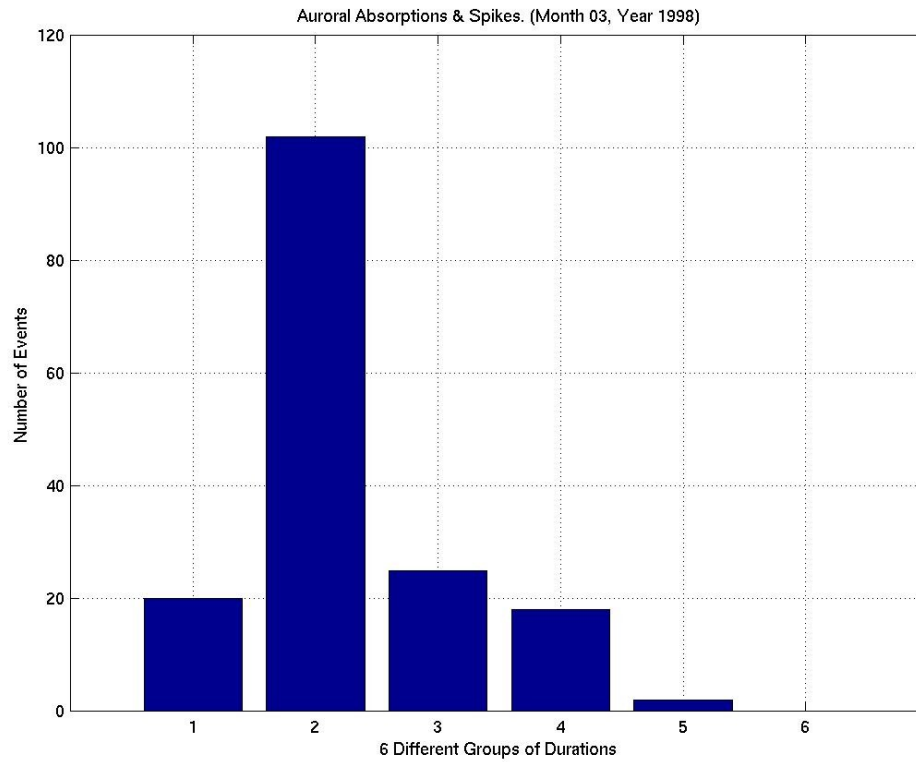


Figure 5.2.42: Statistical Analysis II, Month 03 Year 1998.

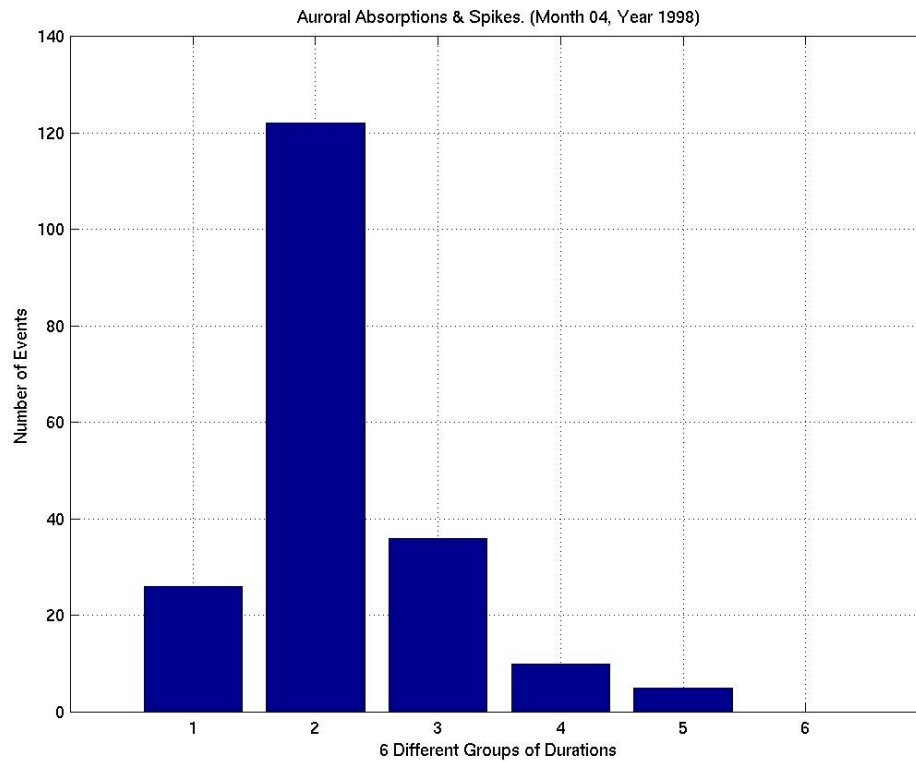


Figure 5.2.43: Statistical Analysis II, Month 04 Year 1998.

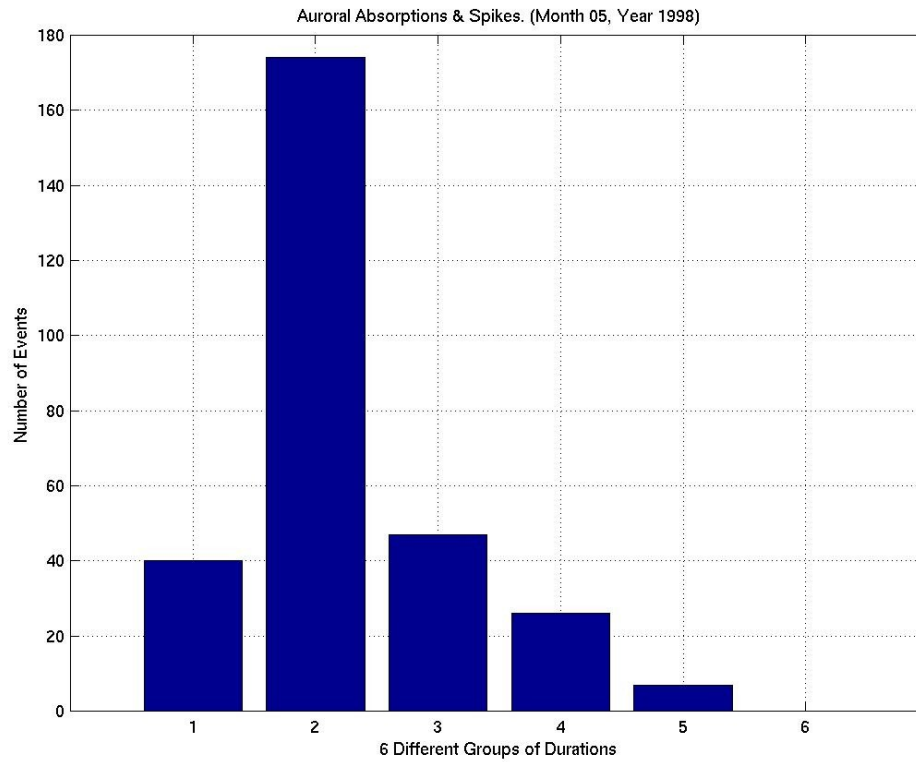


Figure 5.2.44: Statistical Analysis II, Month 05 Year 1998.

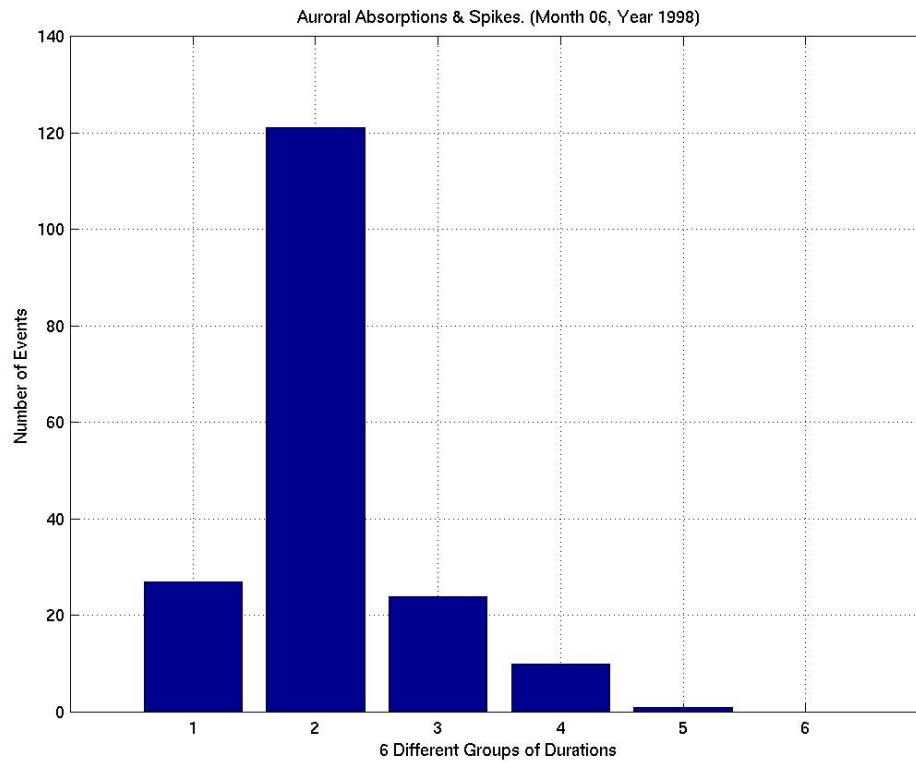


Figure 5.2.45: Statistical Analysis II, Month 06 Year 1998.

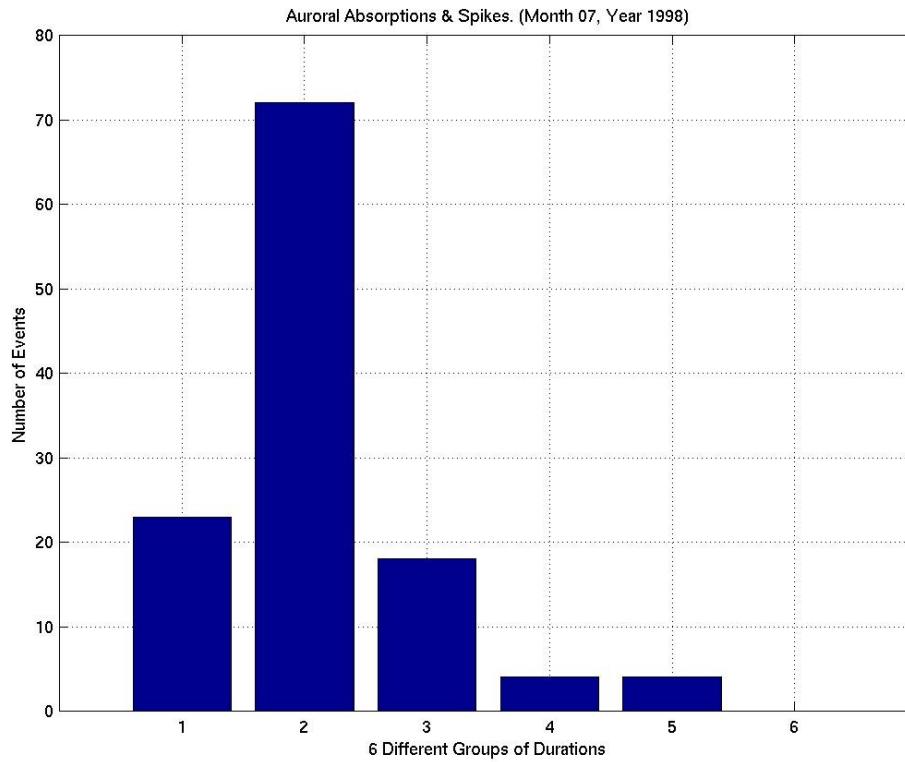


Figure 5.2.46: Statistical Analysis II, Month 07 Year 1998.

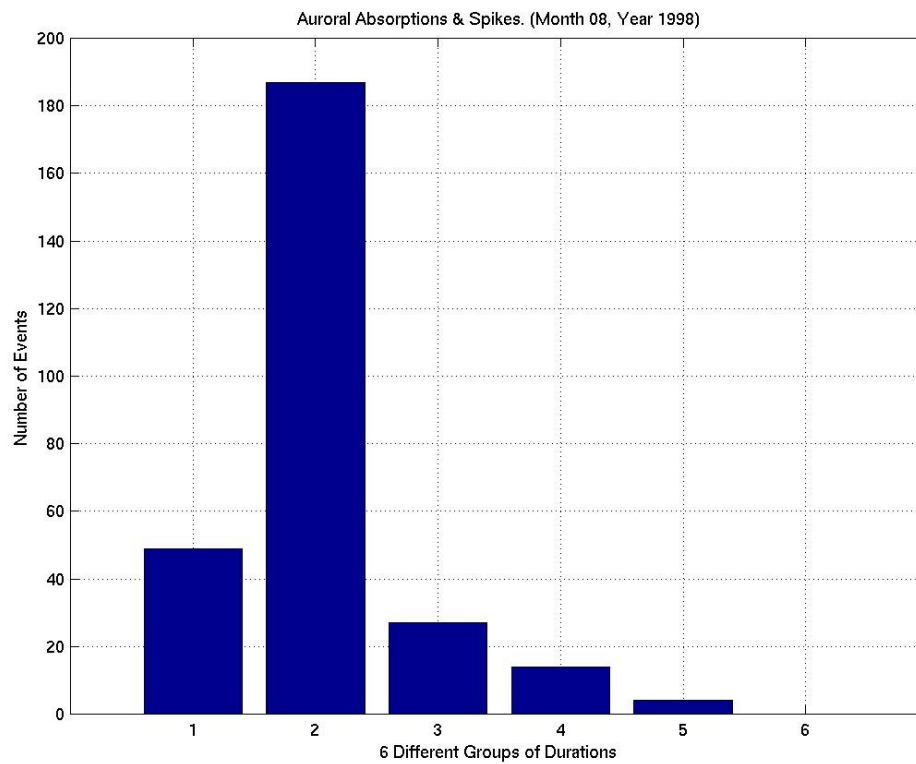


Figure 5.2.47: Statistical Analysis II, Month 08 Year 1998.

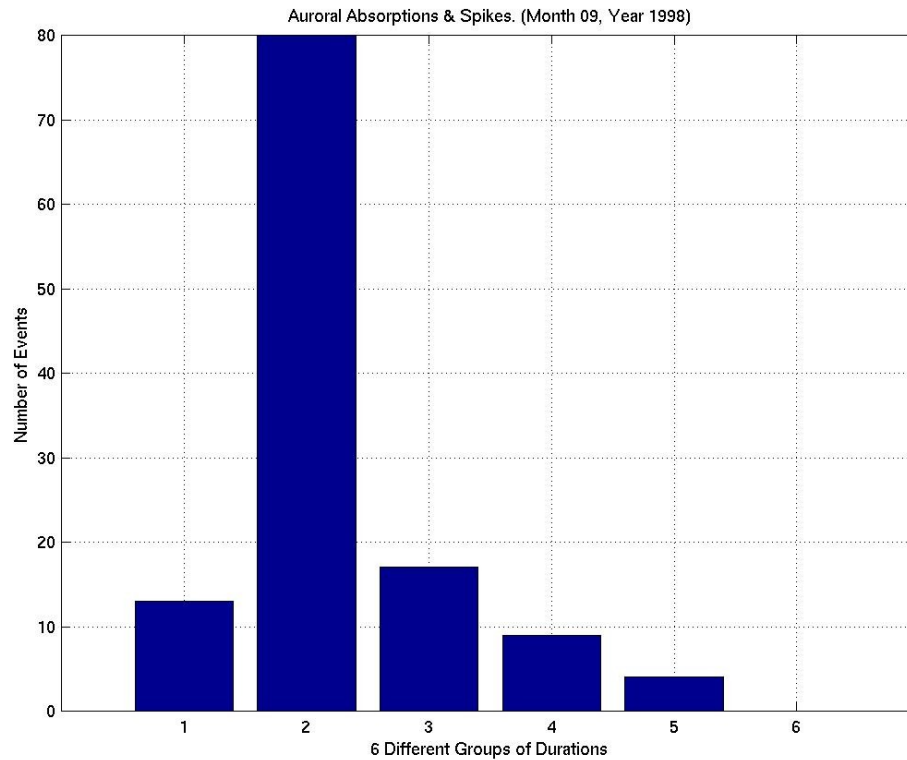


Figure 5.2.48: Statistical Analysis II, Month 09 Year 1998.

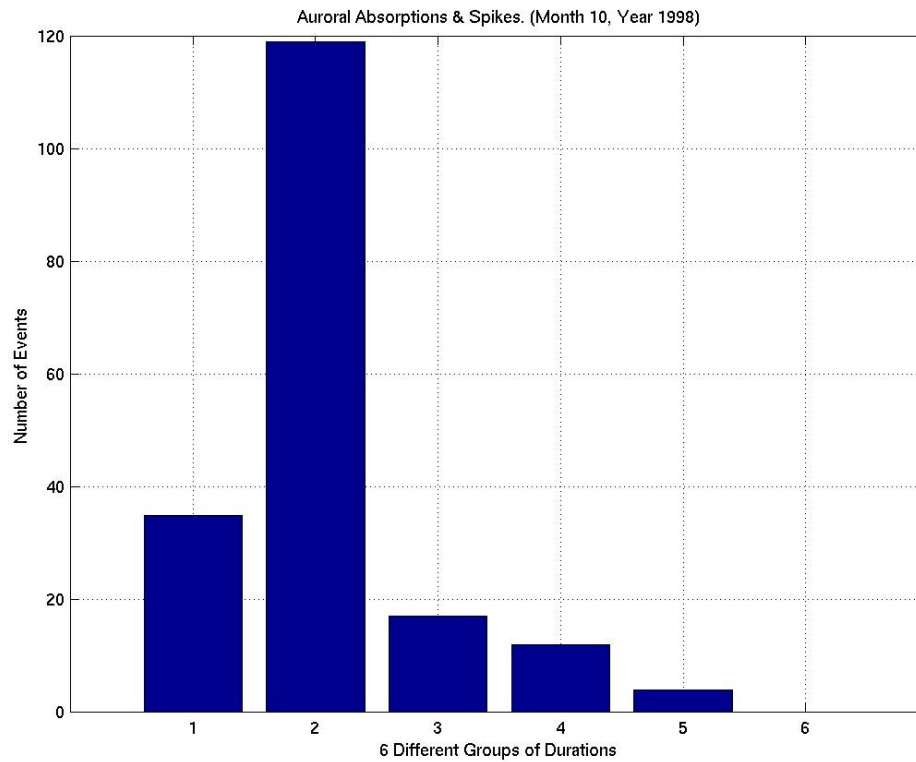


Figure 5.2.49: Statistical Analysis II, Month 10 Year 1998.

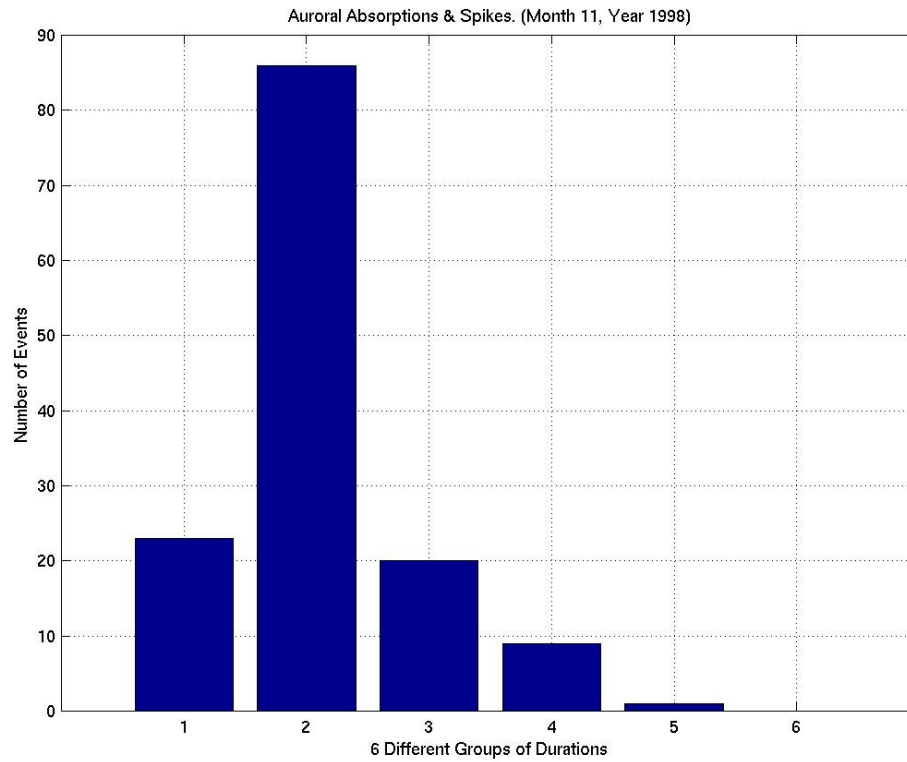


Figure 5.2.50: Statistical Analysis II, Month 11 Year 1998.

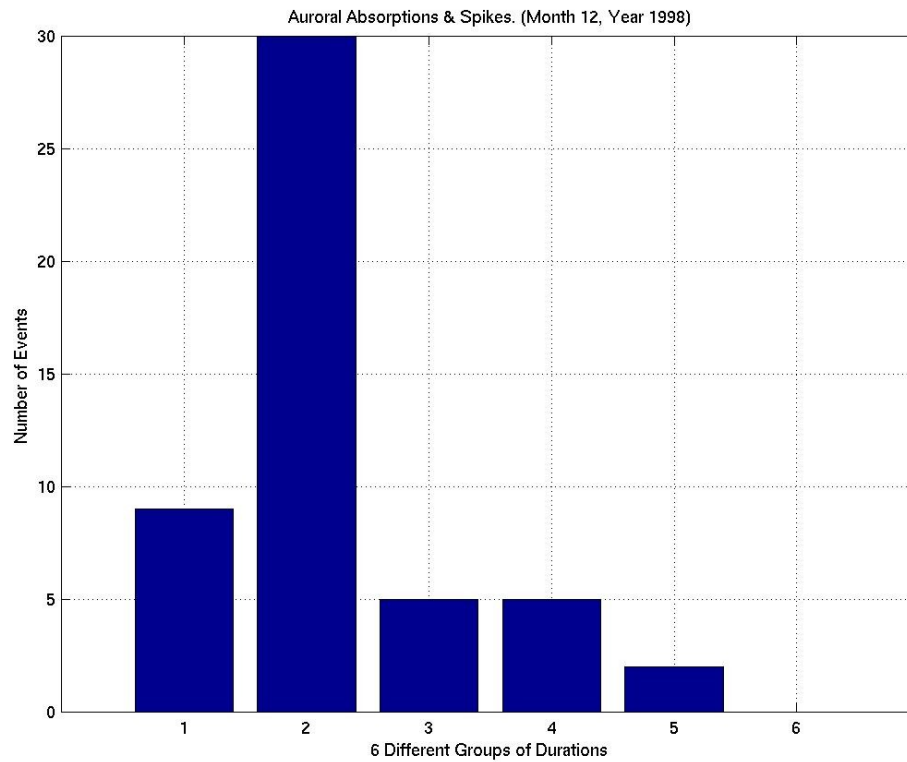


Figure 5.2.51: Statistical Analysis II, Month 12 Year 1998.

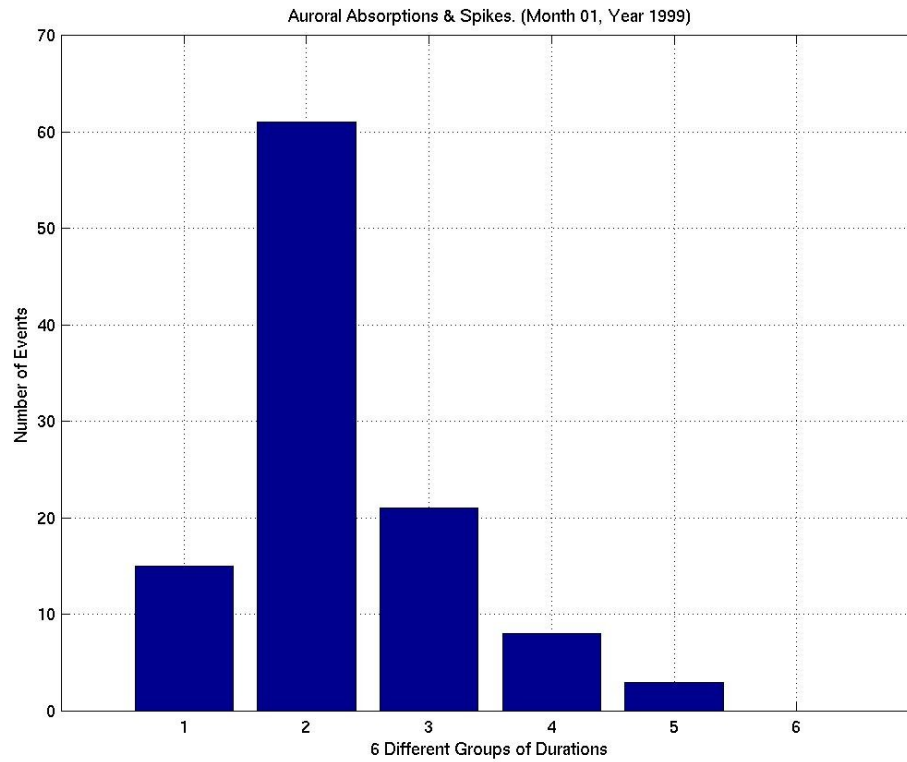


Figure 5.2.52: Statistical Analysis II, Month 01 Year 1999.

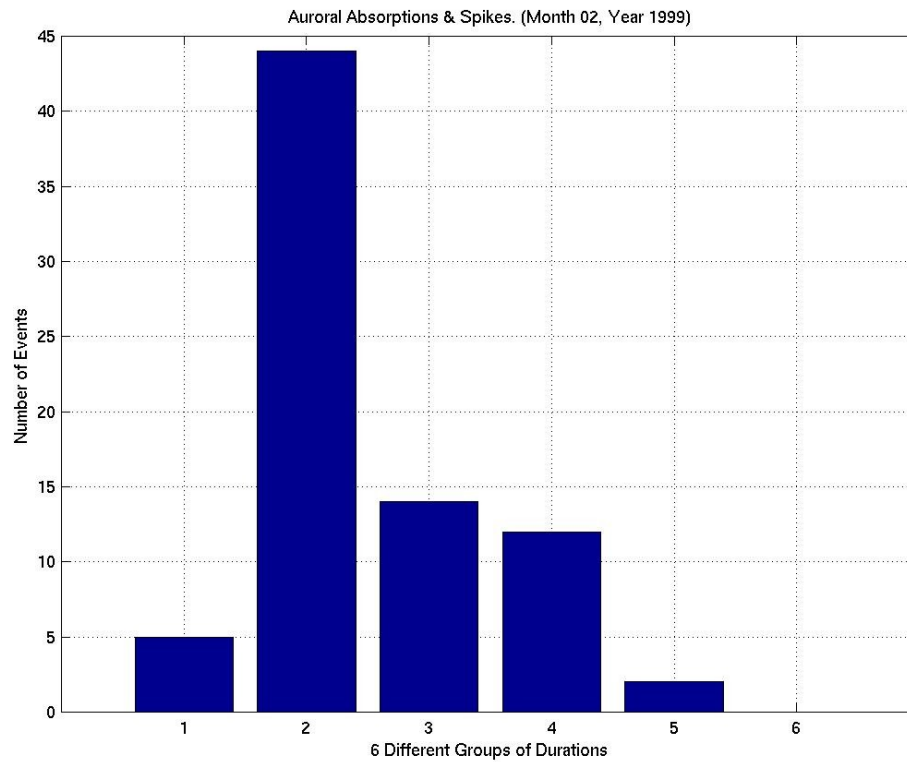


Figure 5.2.53: Statistical Analysis II, Month 02 Year 1999.

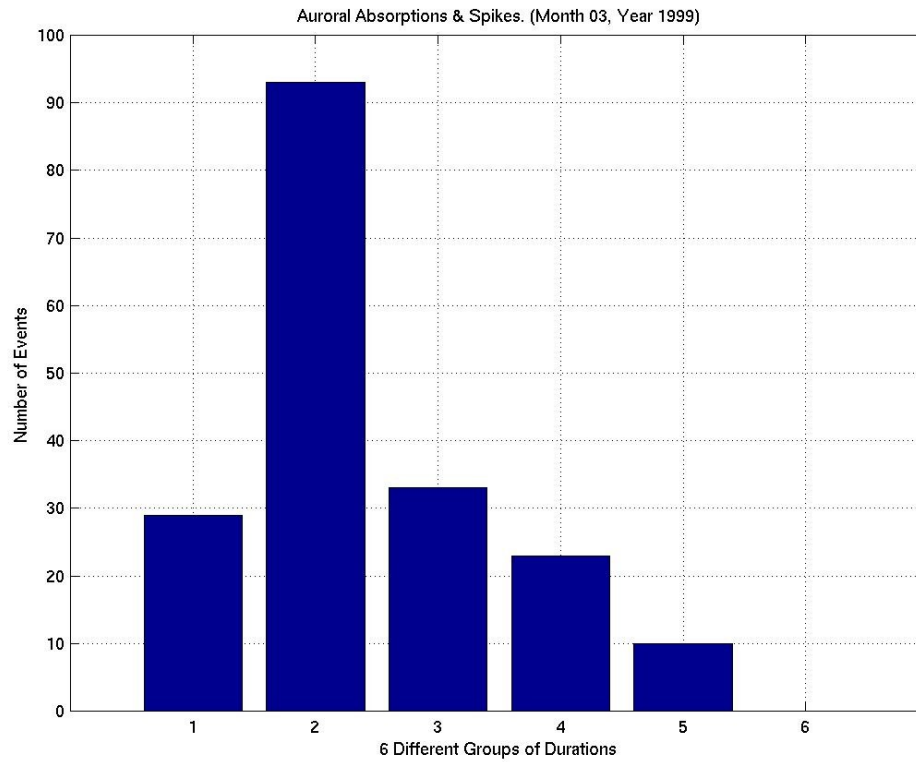


Figure 5.2.54: Statistical Analysis II, Month 03 Year 1999.

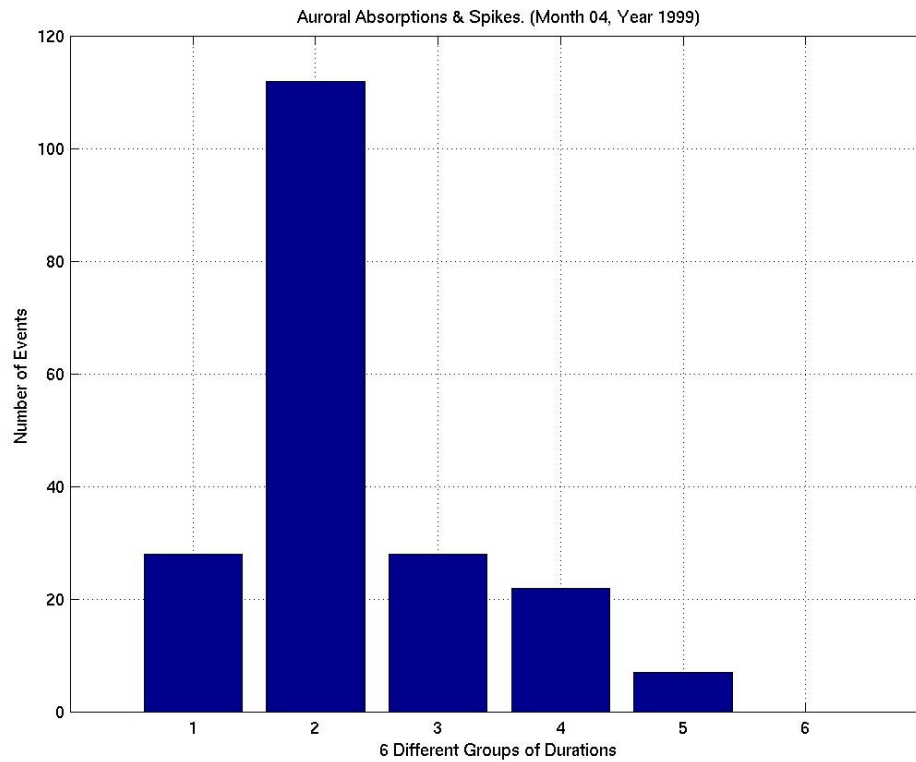


Figure 5.2.55: Statistical Analysis II, Month 04 Year 1999.

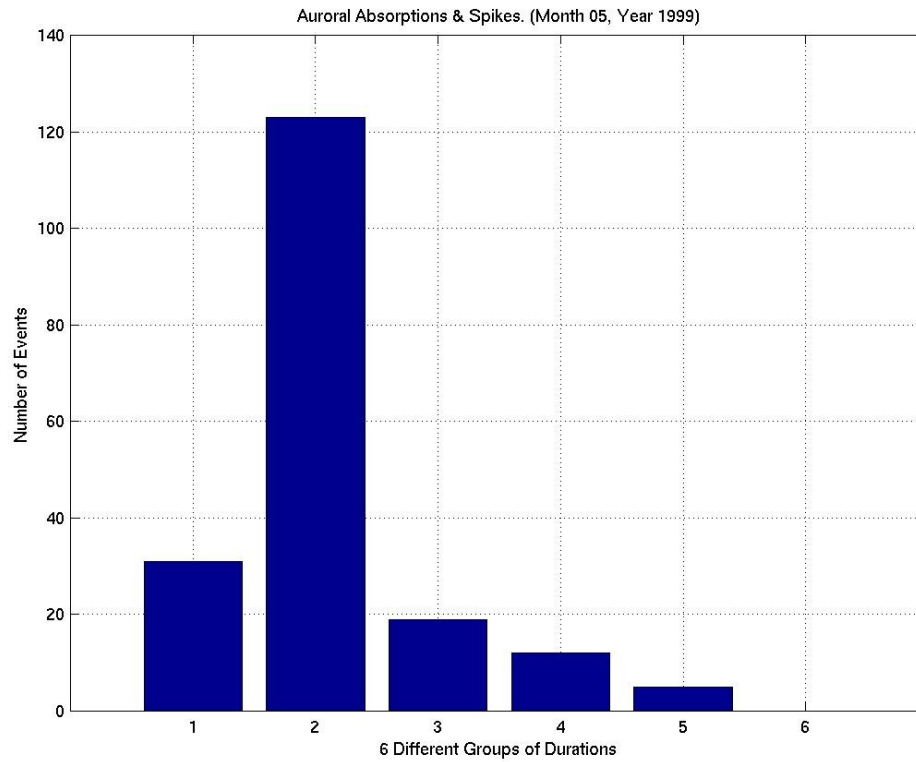


Figure 5.2.56: Statistical Analysis II, Month 05 Year 1999.

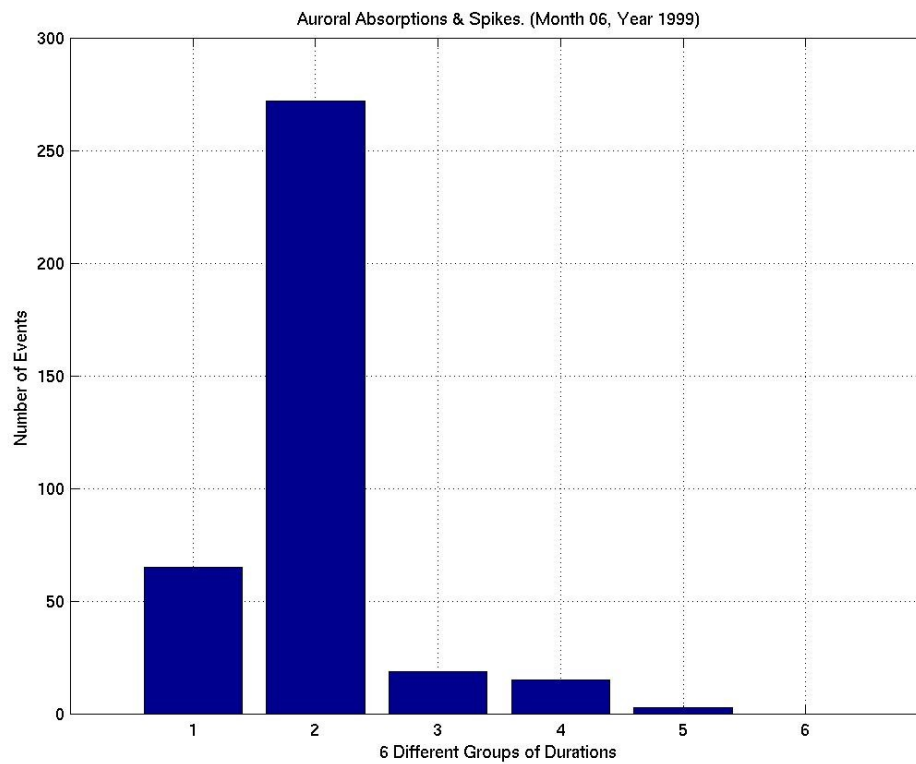


Figure 5.2.57: Statistical Analysis II, Month 06 Year 1999.

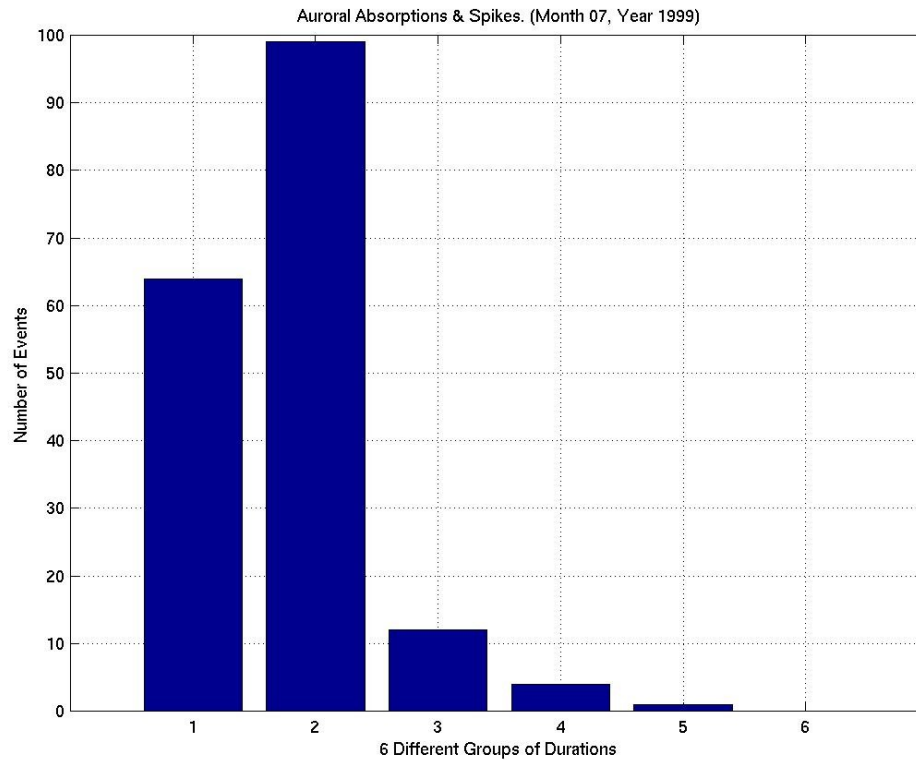


Figure 5.2.58: Statistical Analysis II, Month 07 Year 1999.

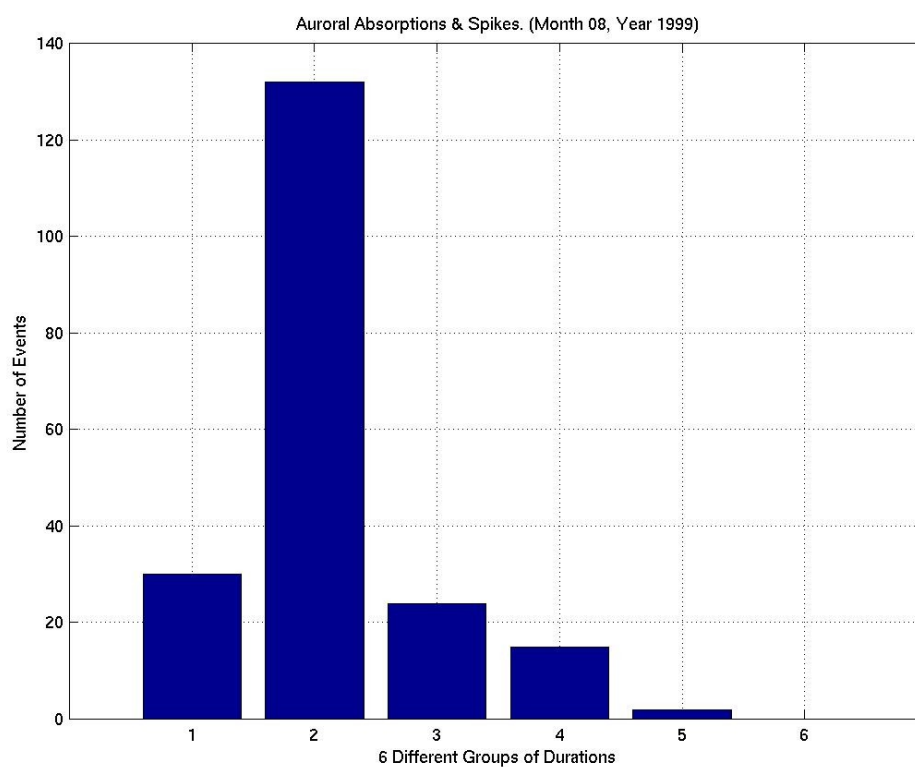


Figure 5.2.59: Statistical Analysis II, Month 08 Year 1999.

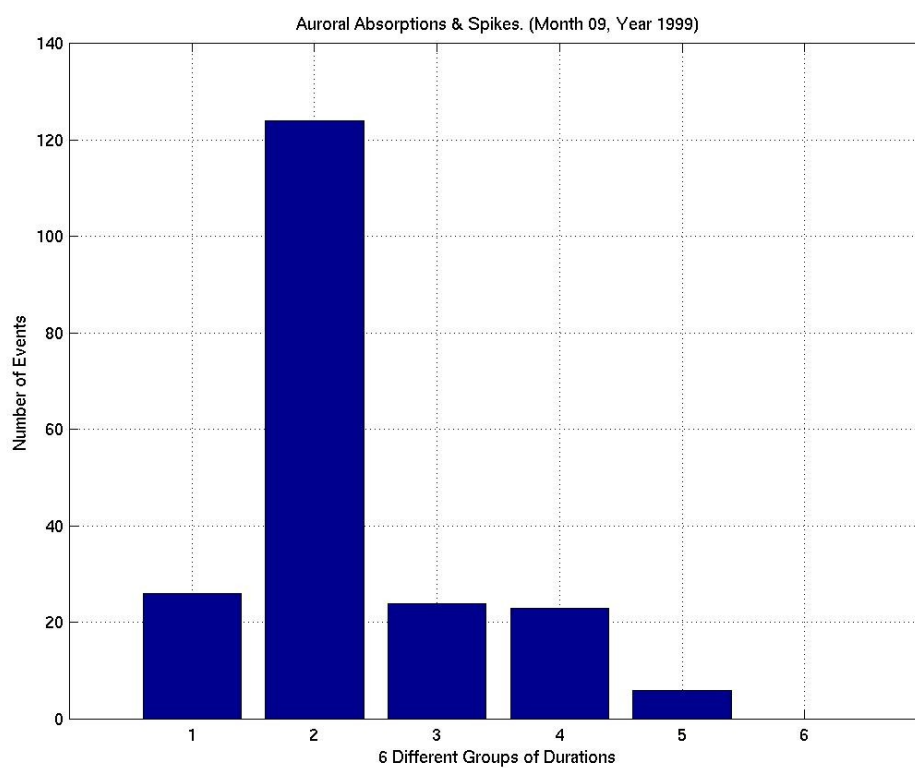


Figure 5.2.60: Statistical Analysis II, Month 09 Year 1999.

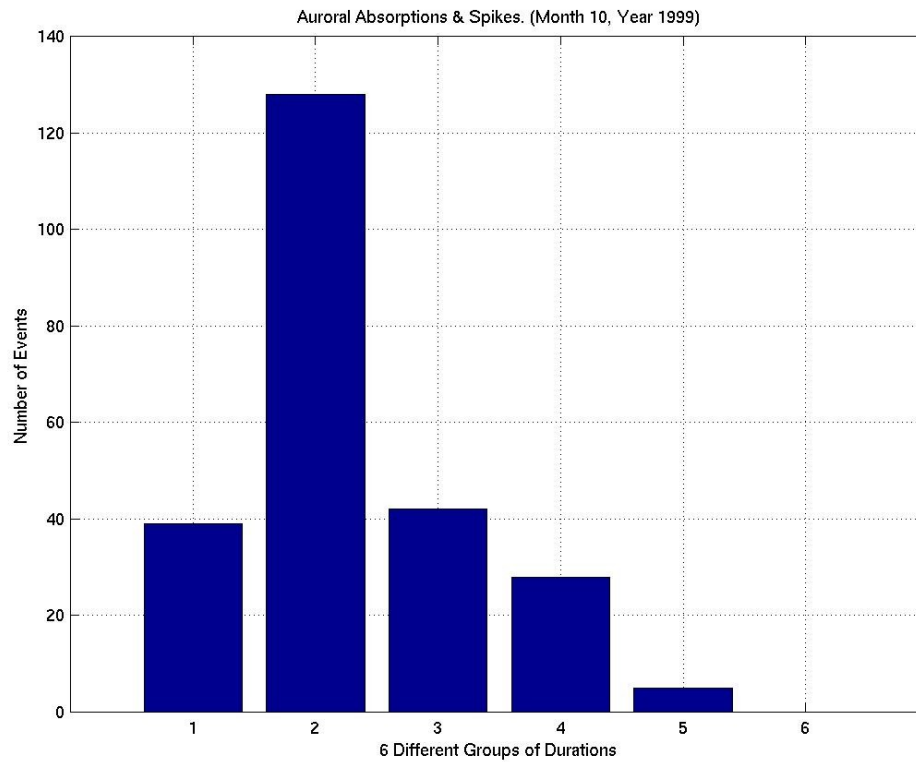


Figure 5.2.61: Statistical Analysis II, Month 10 Year 1999.

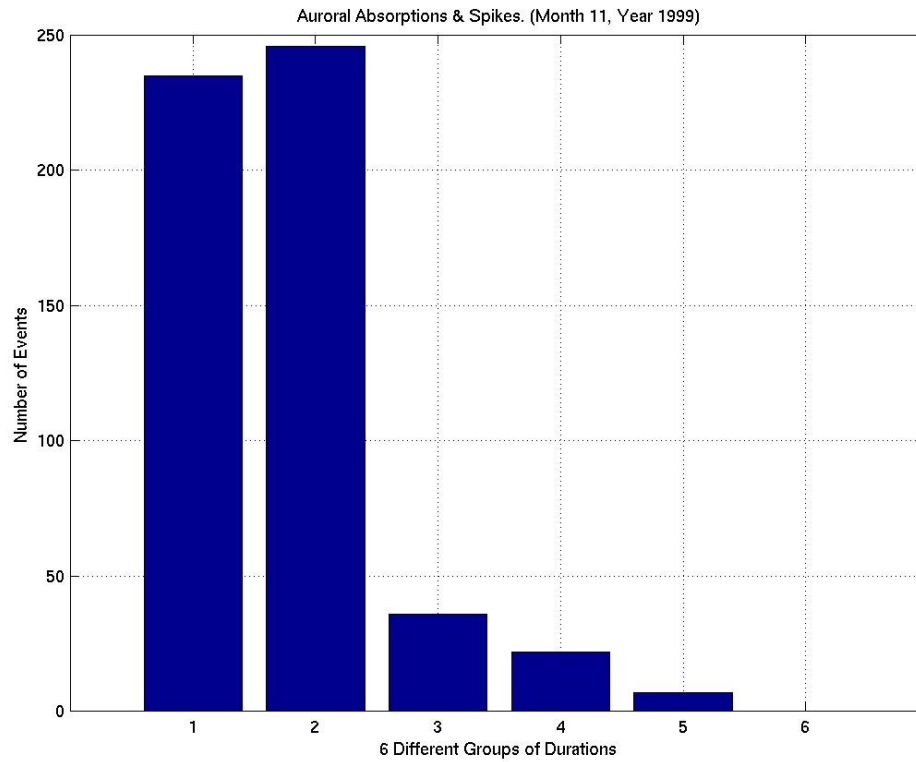


Figure 5.2.62: Statistical Analysis II, Month 11 Year 1999.

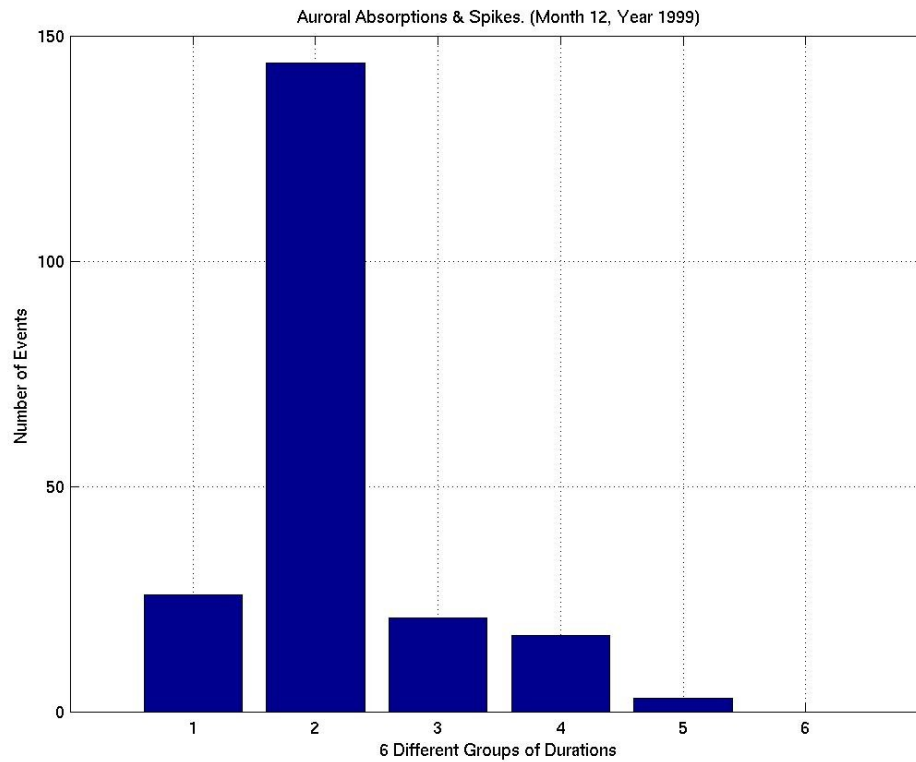


Figure 5.2.63: Statistical Analysis II, Month 12 Year 1999.

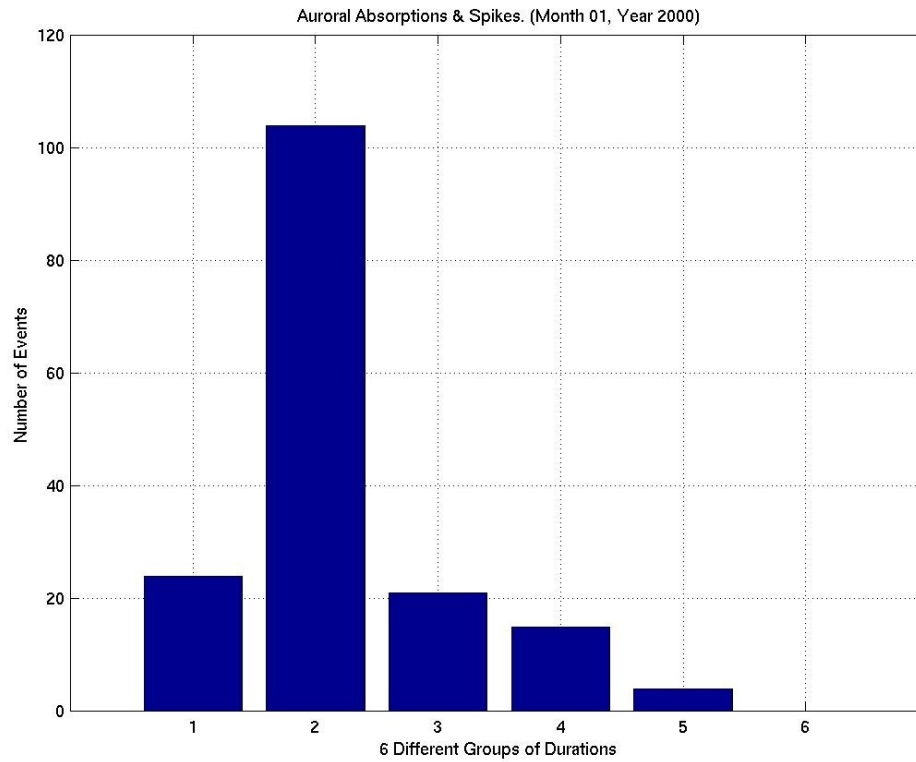


Figure 5.2.64: Statistical Analysis II, Month 01 Year 2000.

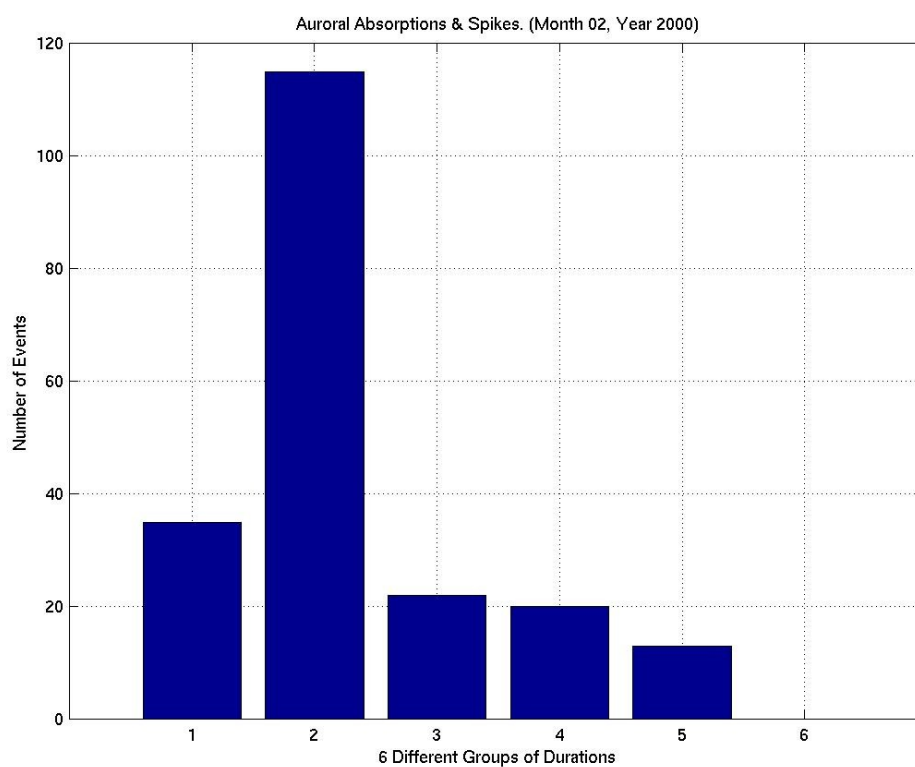


Figure 5.2.65: Statistical Analysis II, Month 02 Year 2000.

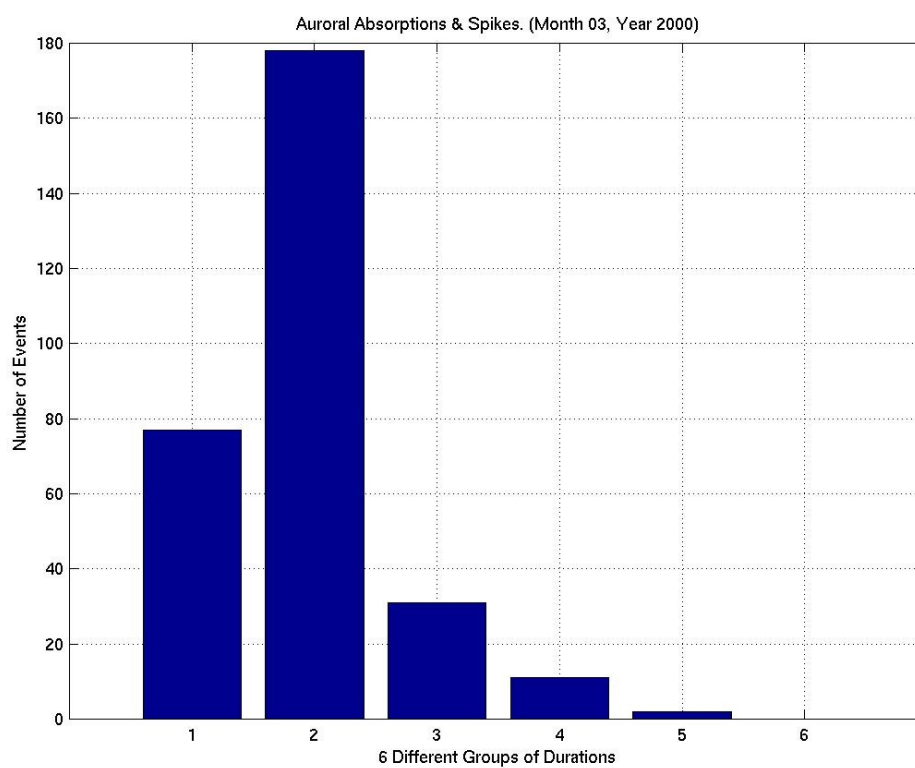


Figure 5.2.66: Statistical Analysis II, Month 03 Year 2000.

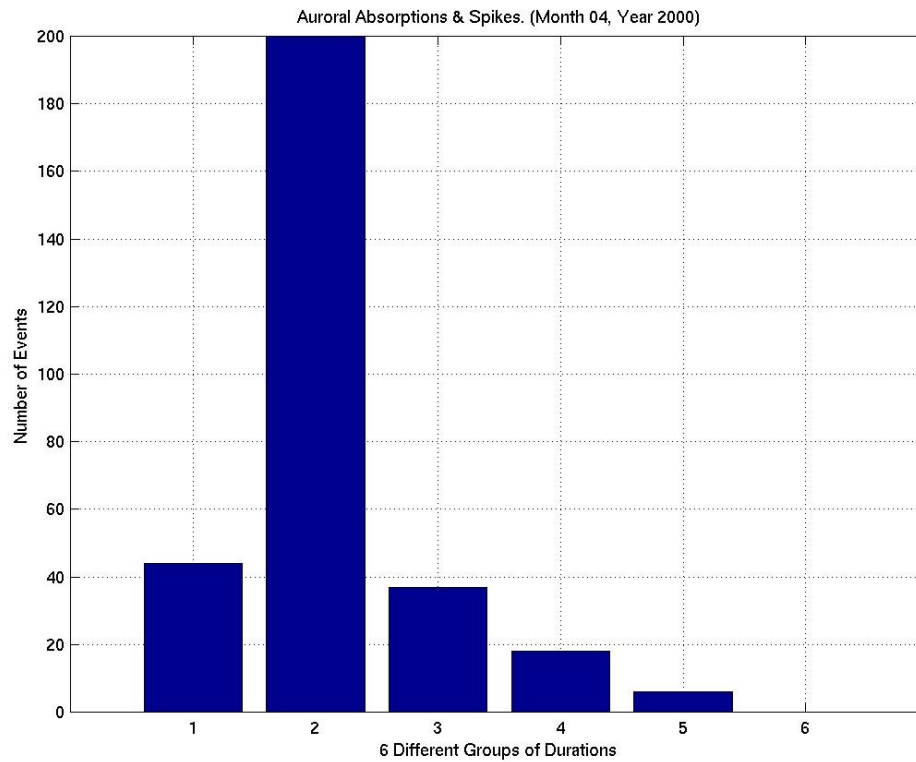


Figure 5.2.67: Statistical Analysis II, Month 04 Year 2000.

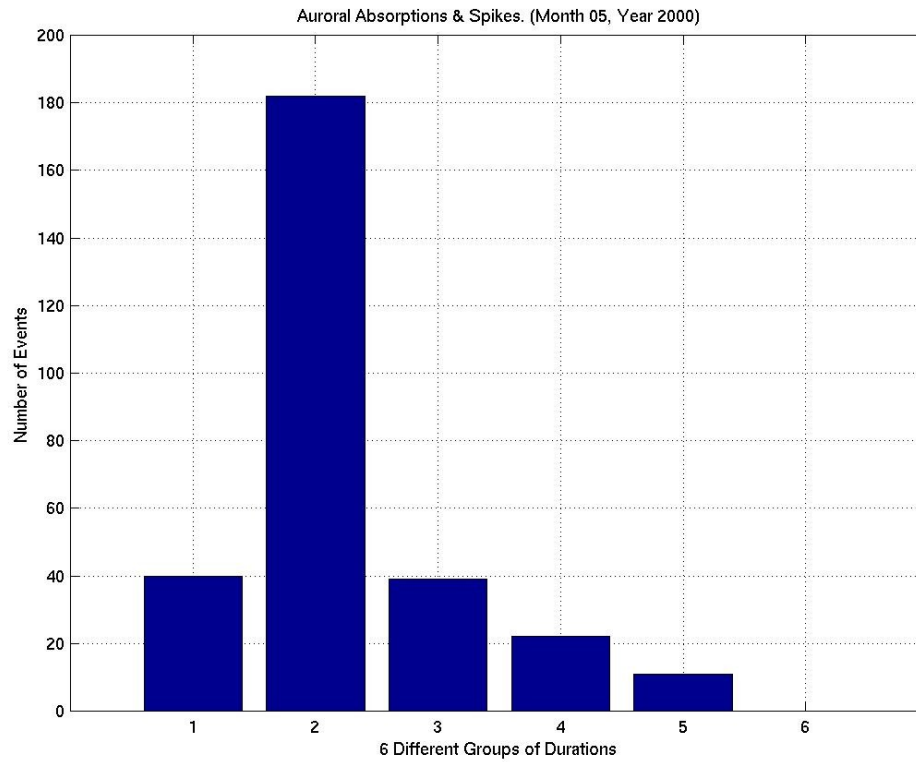


Figure 5.2.68: Statistical Analysis II, Month 05 Year 2000.

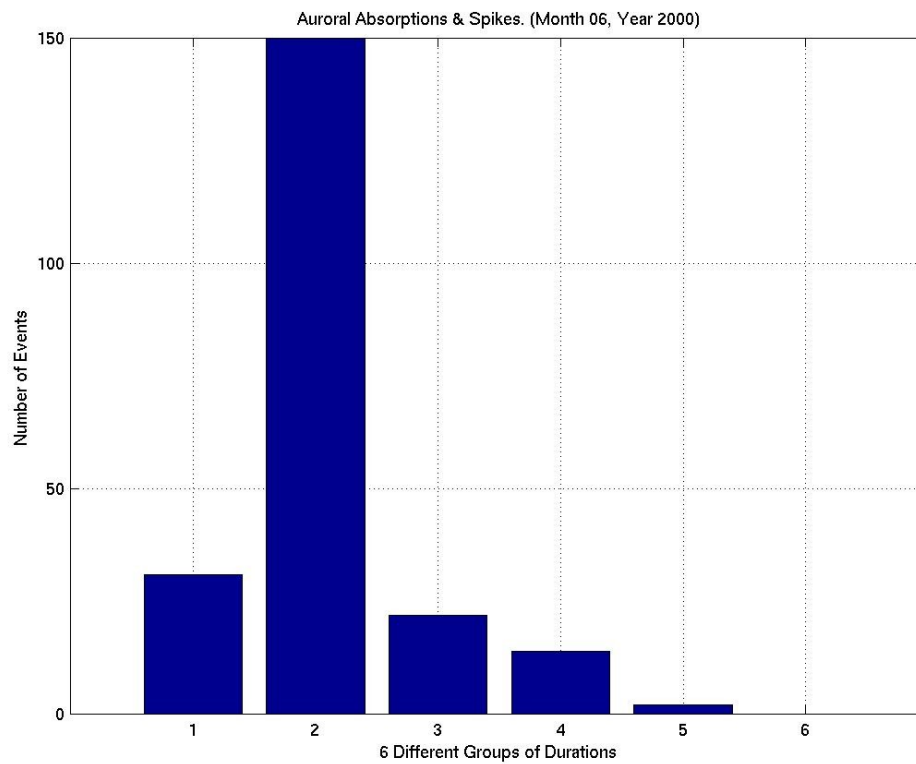


Figure 5.2.69: Statistical Analysis II, Month 06 Year 2000.

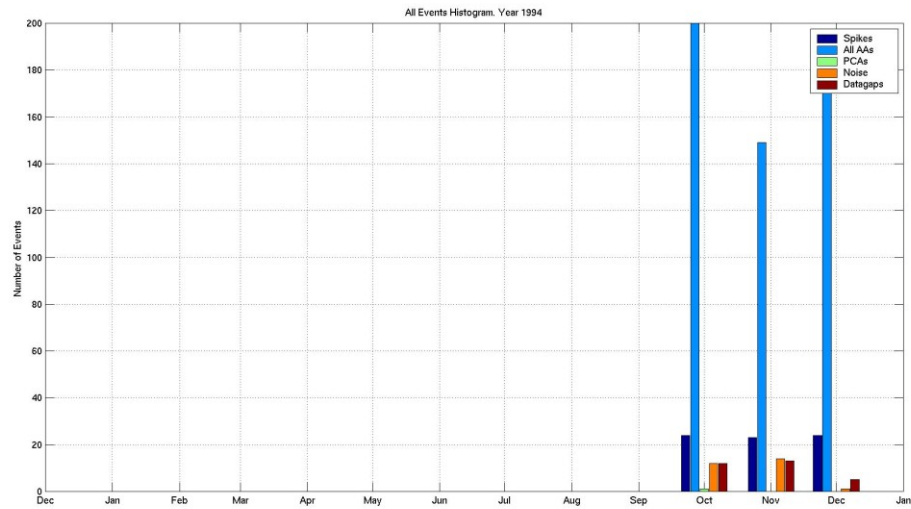


Figure 5.3.1: Statistical Analysis III, Year 1994.

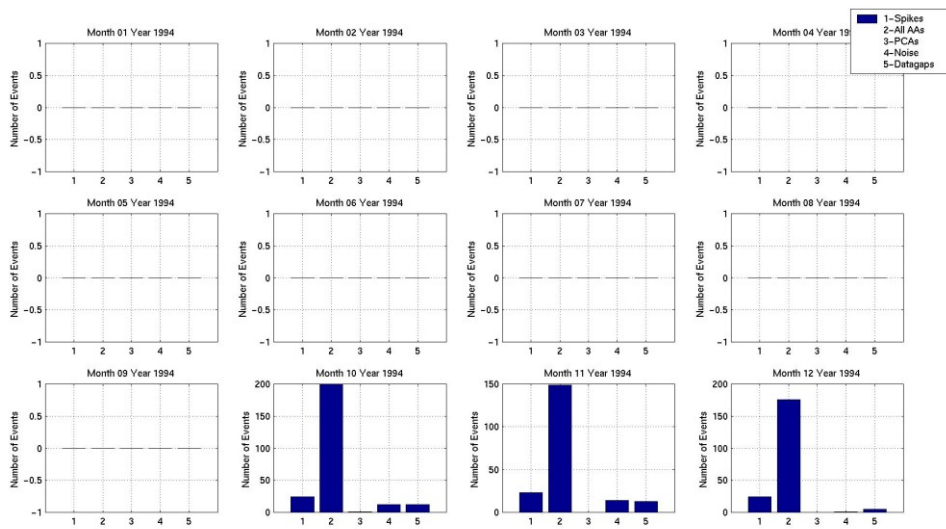


Figure 5.3.1b: Statistical Analysis III, Year 1994.

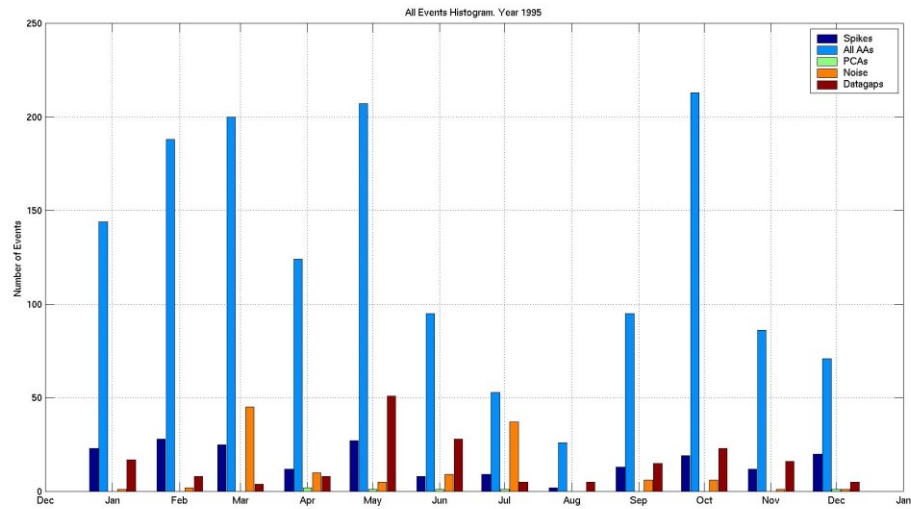


Figure 5.3.2: Statistical Analysis III, Year 1995.

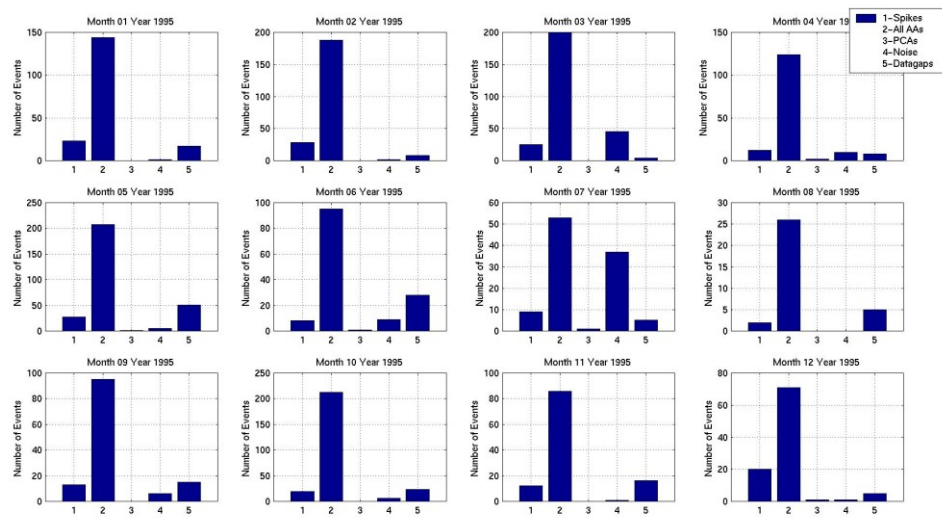


Figure 5.3.2b: Statistical Analysis III, Year 1995.

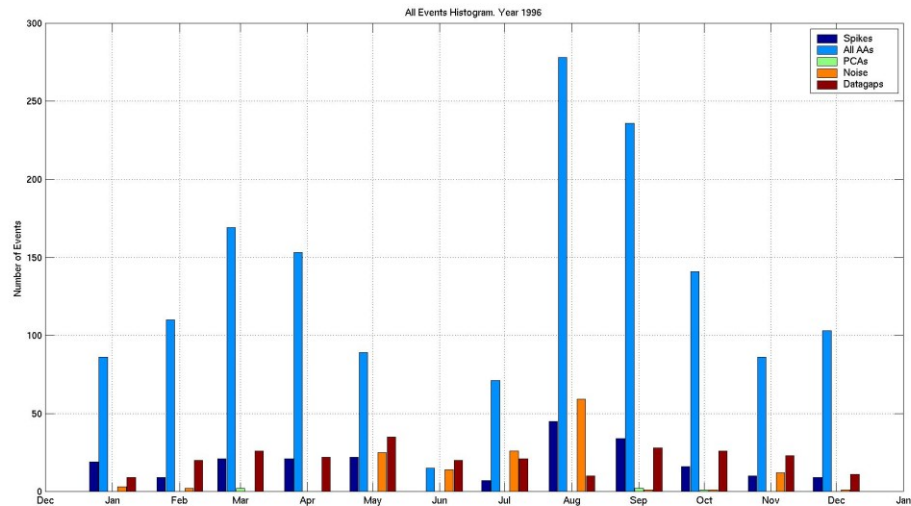


Figure 5.3.3: Statistical Analysis III, Year 1996.

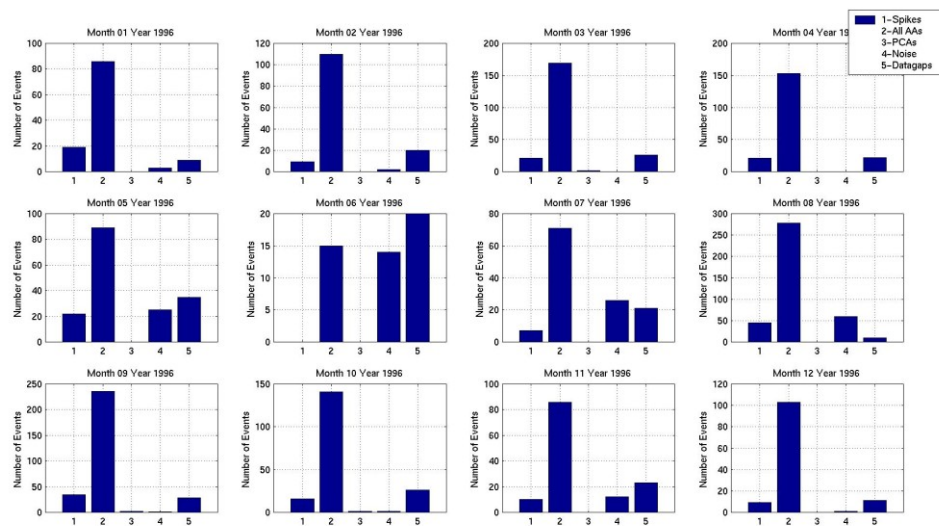


Figure 5.3.3b: Statistical Analysis III, Year 1996.

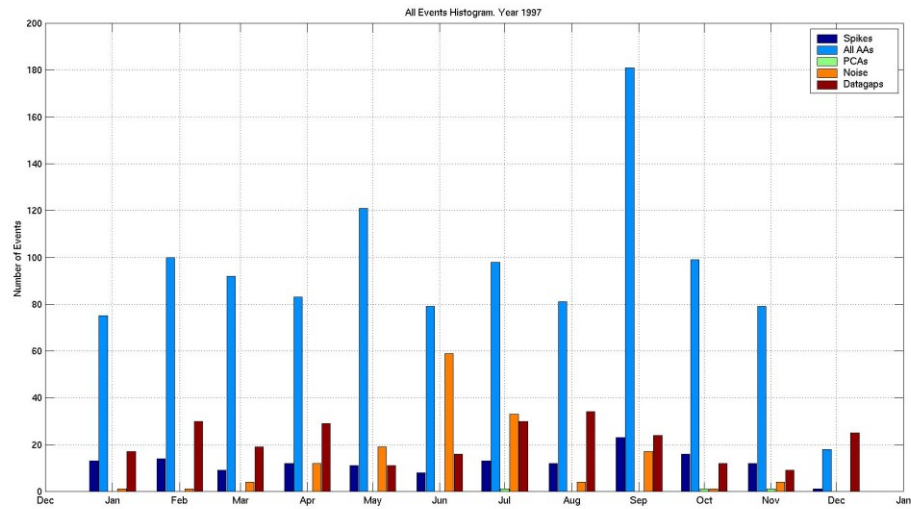


Figure 5.3.4: Statistical Analysis III, Year 1997.

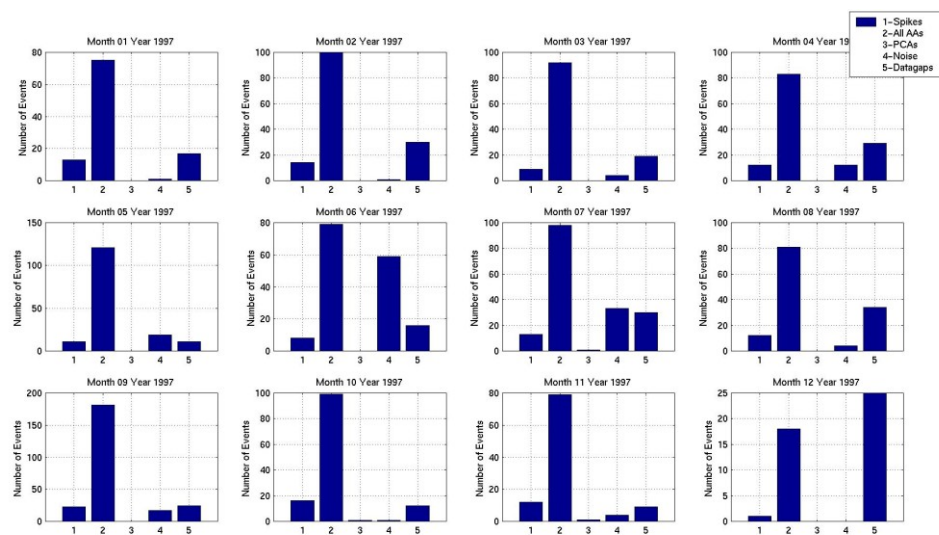


Figure 5.3.4b: Statistical Analysis III, Year 1997.

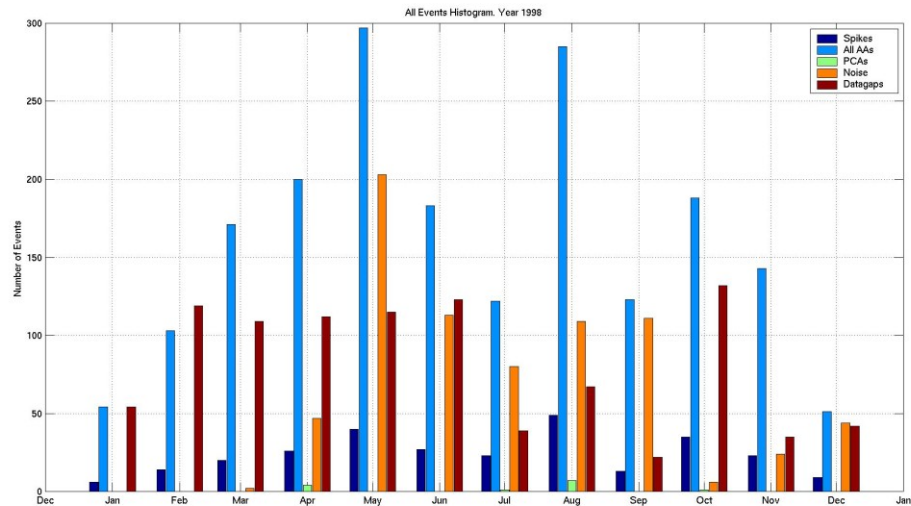


Figure 5.3.5: Statistical Analysis III, Year 1998.

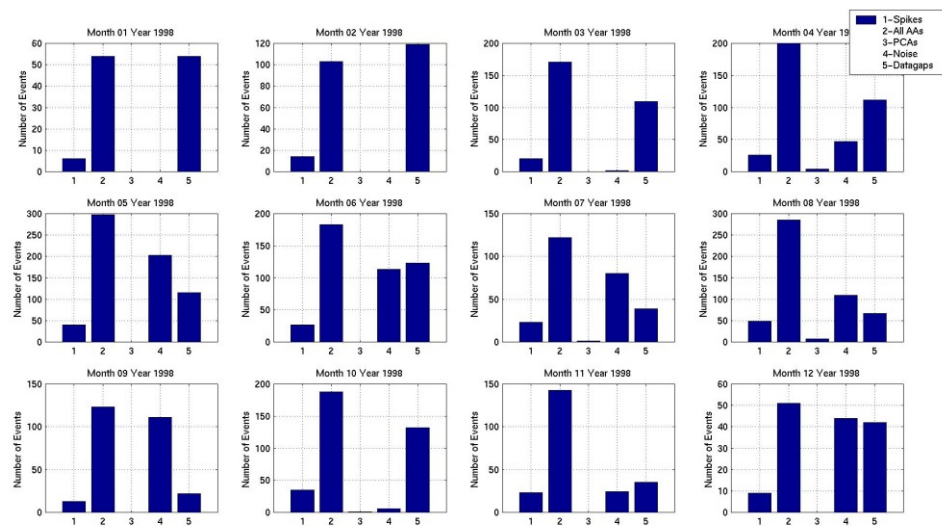


Figure 5.3.5b: Statistical Analysis III, Year 1998.

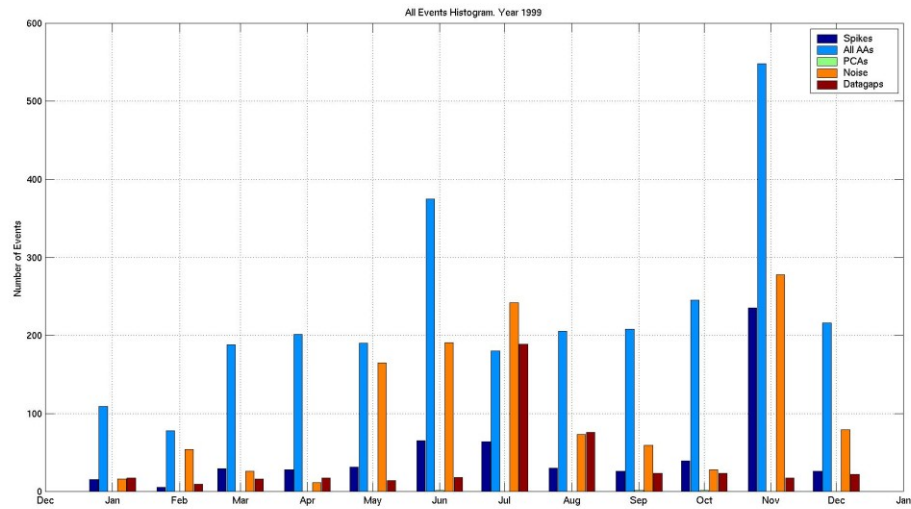


Figure 5.3.6: Statistical Analysis III, Year 1999.

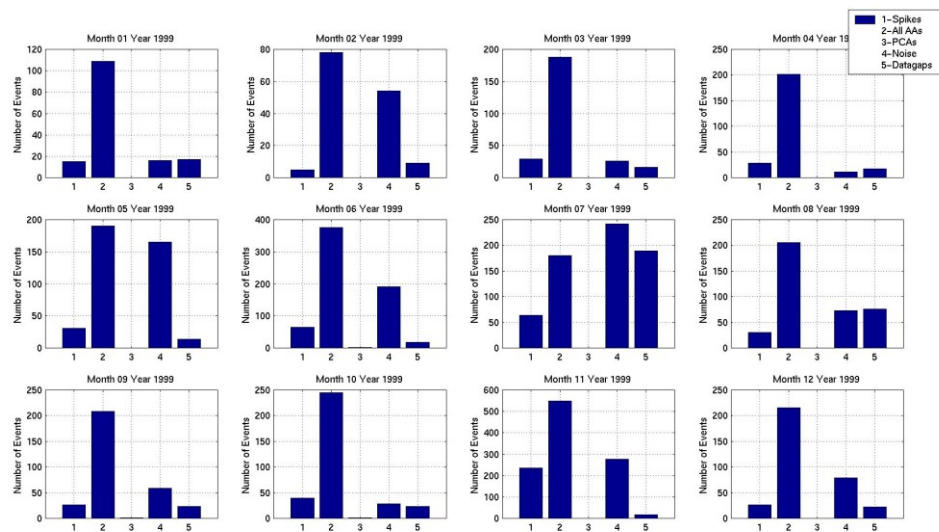


Figure 5.3.6b: Statistical Analysis III, Year 1999.

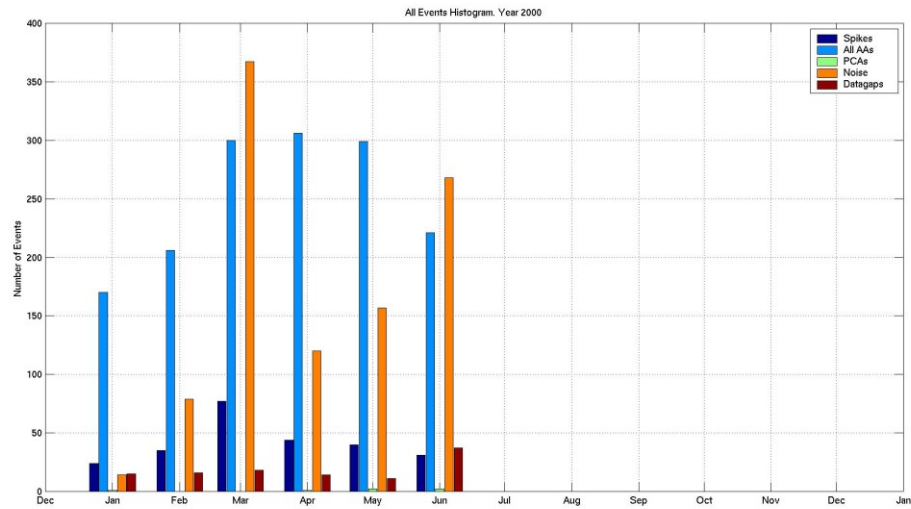


Figure 5.3.7: Statistical Analysis III, Year 2000.

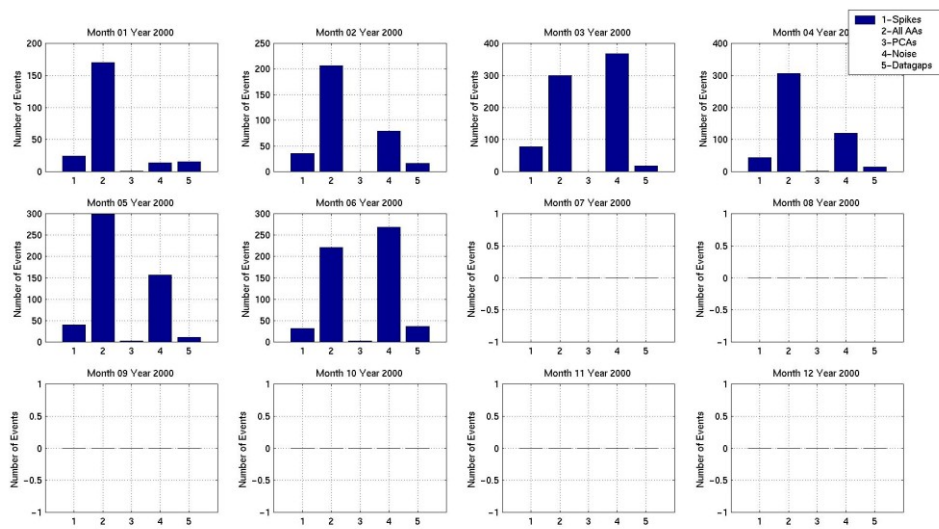


Figure 5.3.7b: Statistical Analysis III, Year 2000.

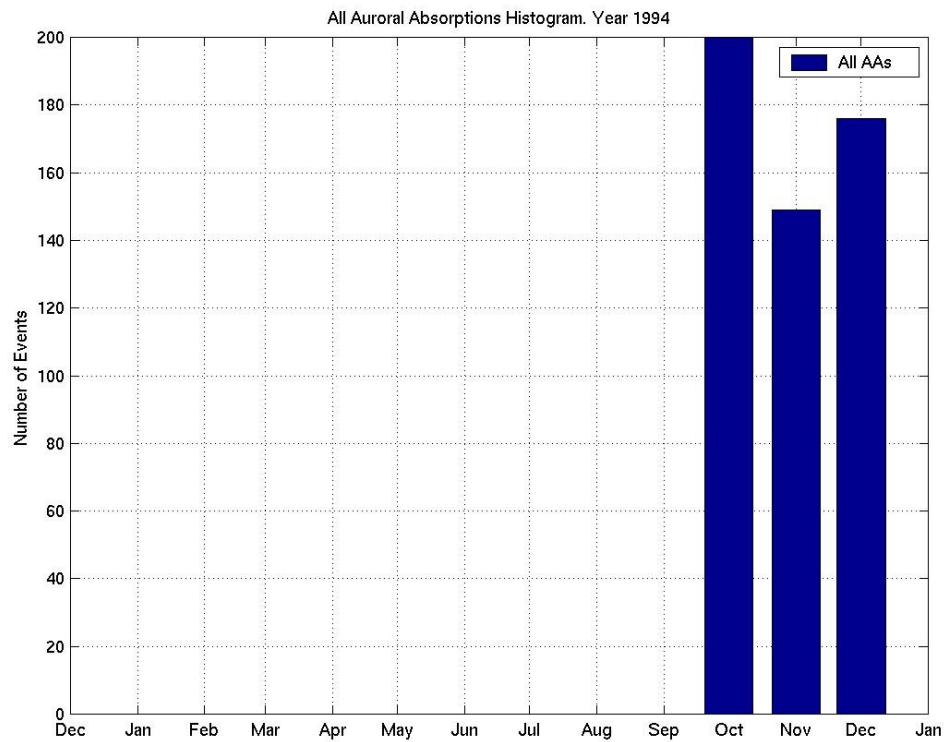


Figure 5.3.1c-1: Statistical Analysis III, Year 1994.

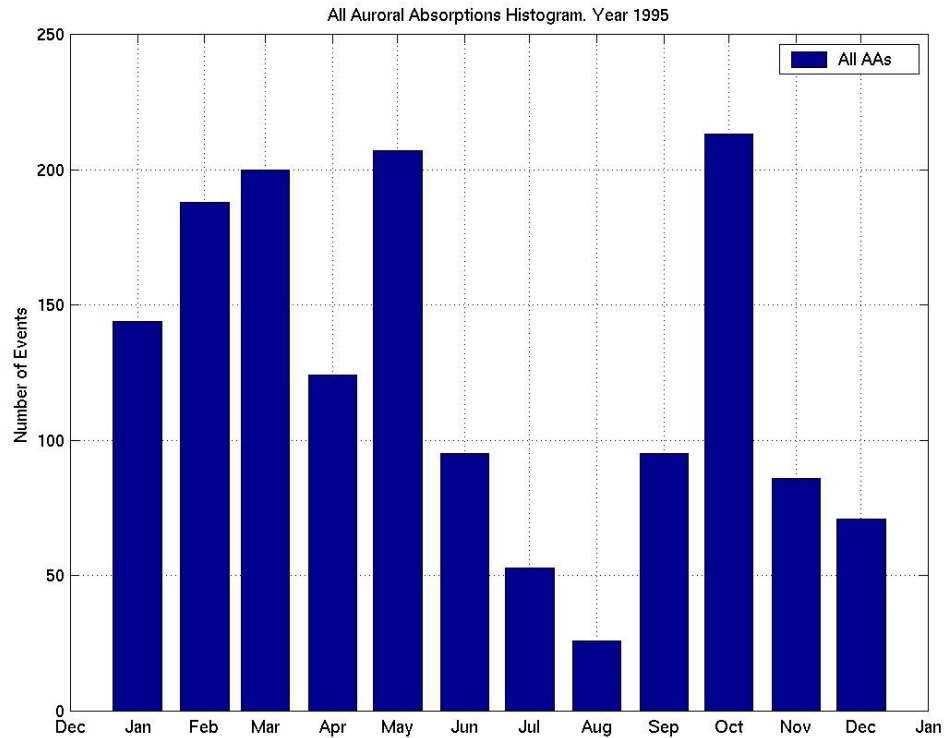


Figure 5.3.2c-1: Statistical Analysis III, Year 1995.

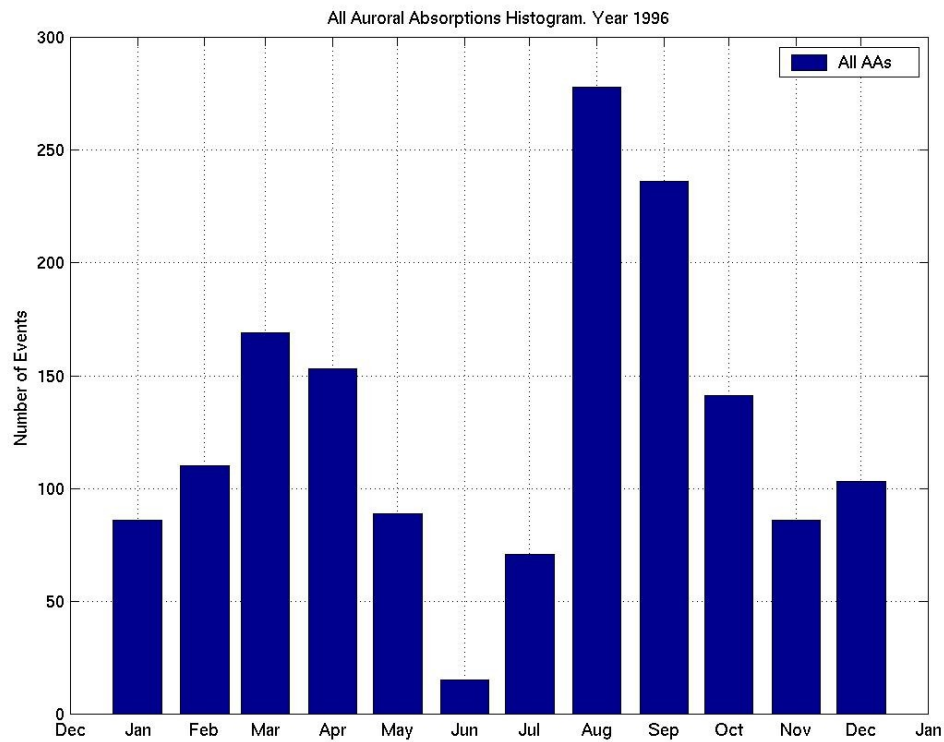


Figure 5.3.3c-1: Statistical Analysis III, Year 1996.

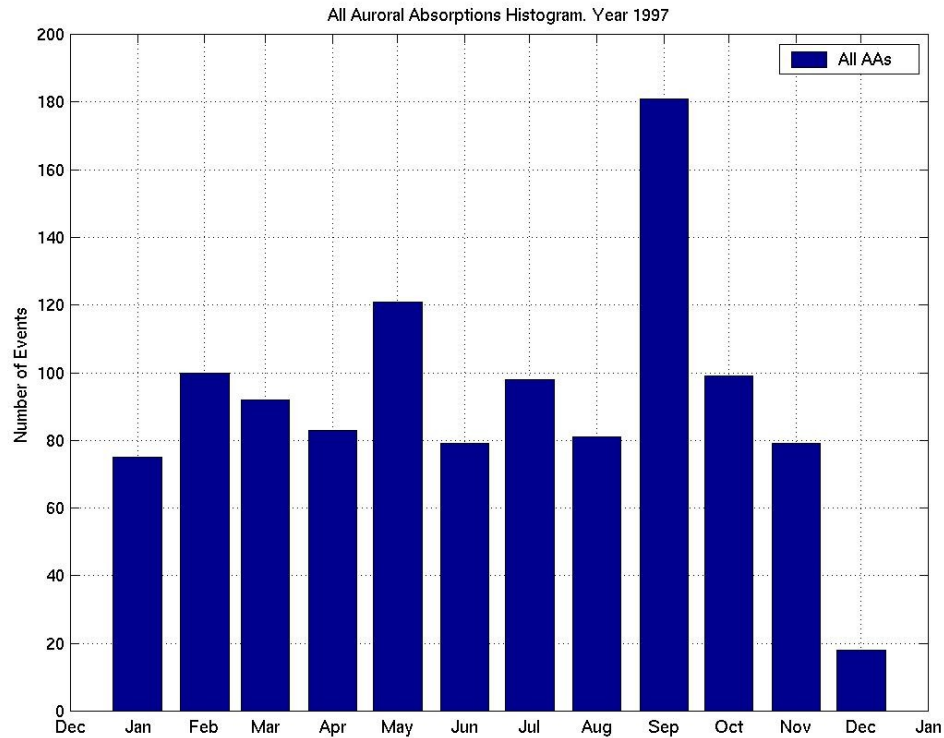


Figure 5.3.4c-1: Statistical Analysis III, Year 1997.

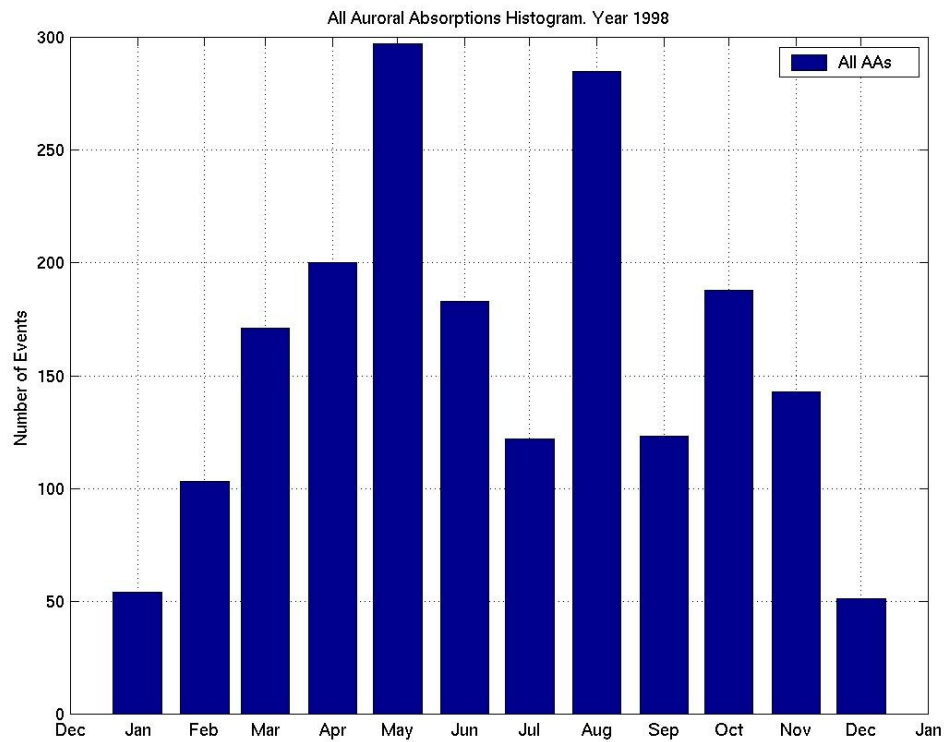


Figure 5.3.5c-1: Statistical Analysis III, Year 1998.

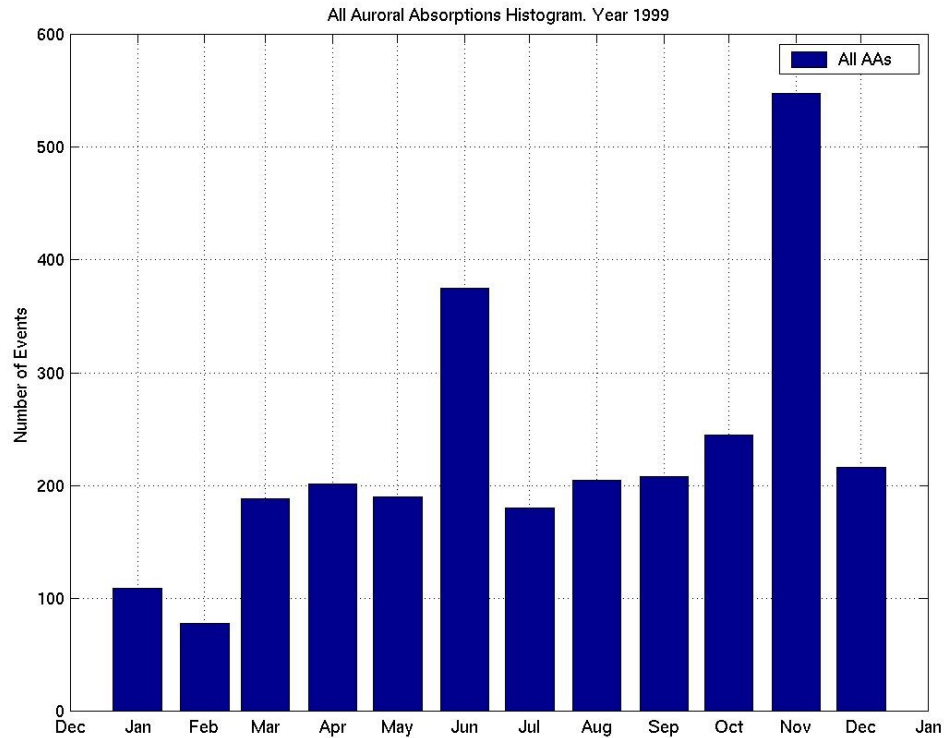


Figure 5.3.6c-1: Statistical Analysis III, Year 1999.

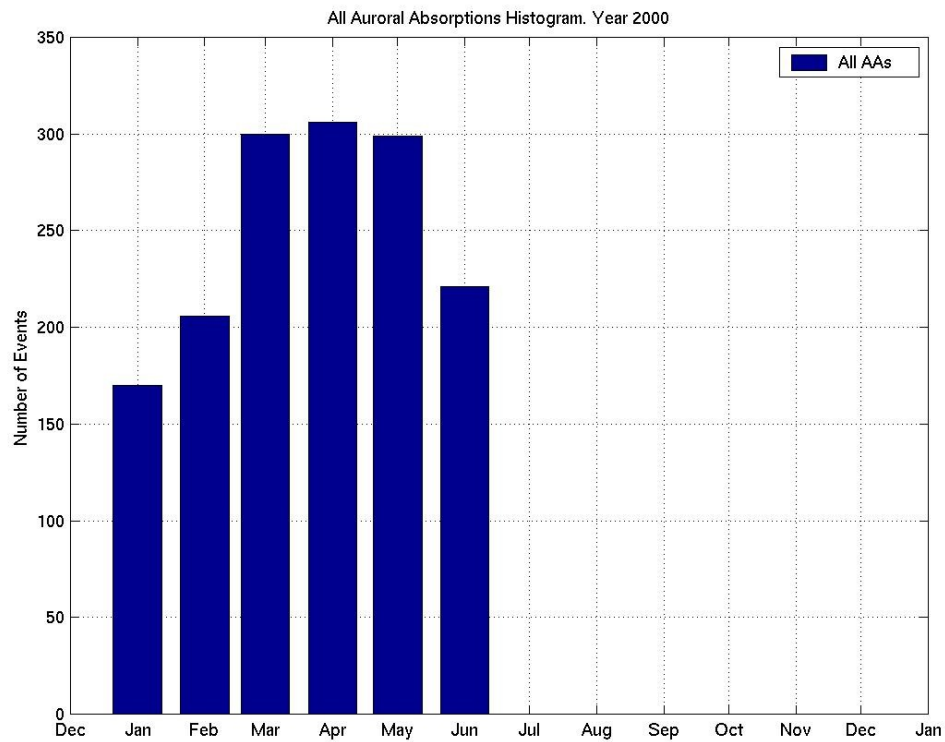


Figure 5.3.7c-1: Statistical Analysis III, Year 2000.

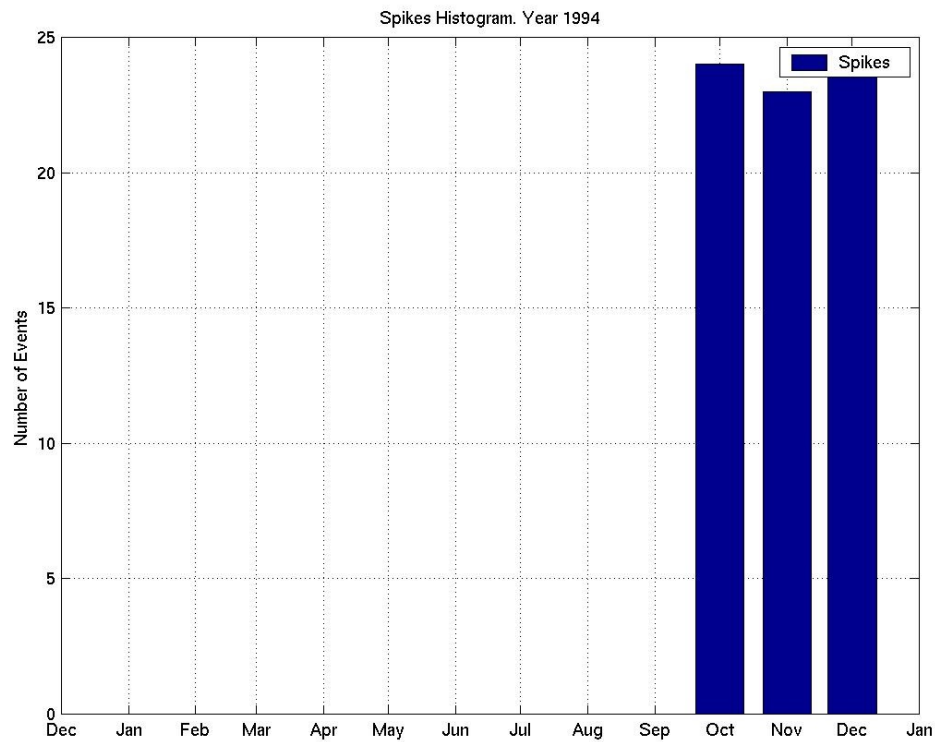


Figure 5.3.1c-2: Statistical Analysis III, Year 1994.

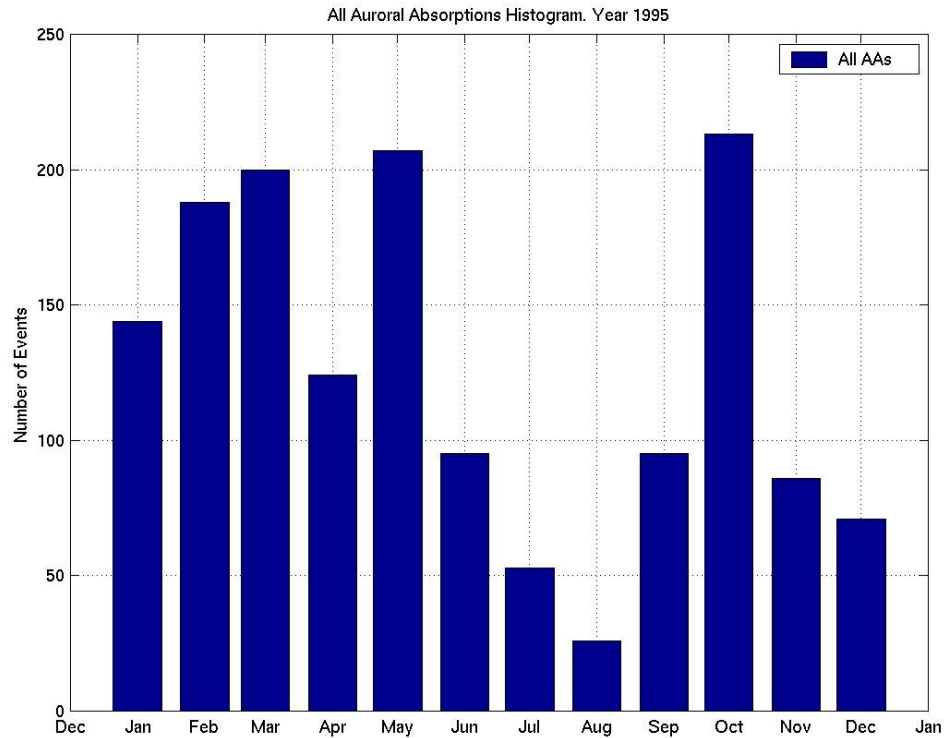


Figure 5.3.2c-2: Statistical Analysis III, Year 1995.

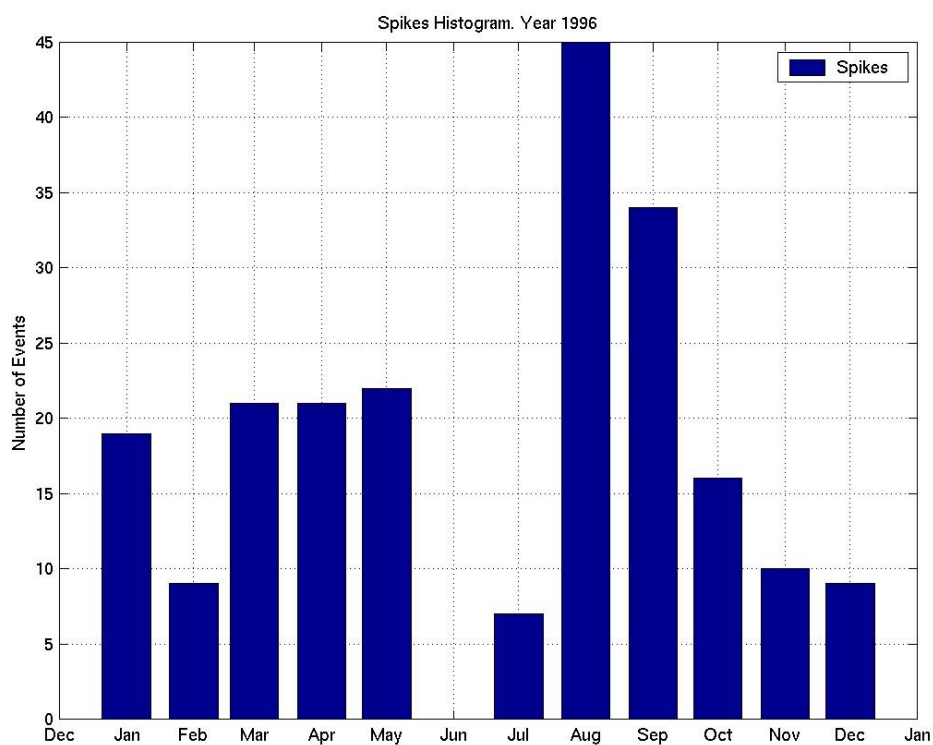


Figure 5.3.3c-2: Statistical Analysis III, Year 1996.

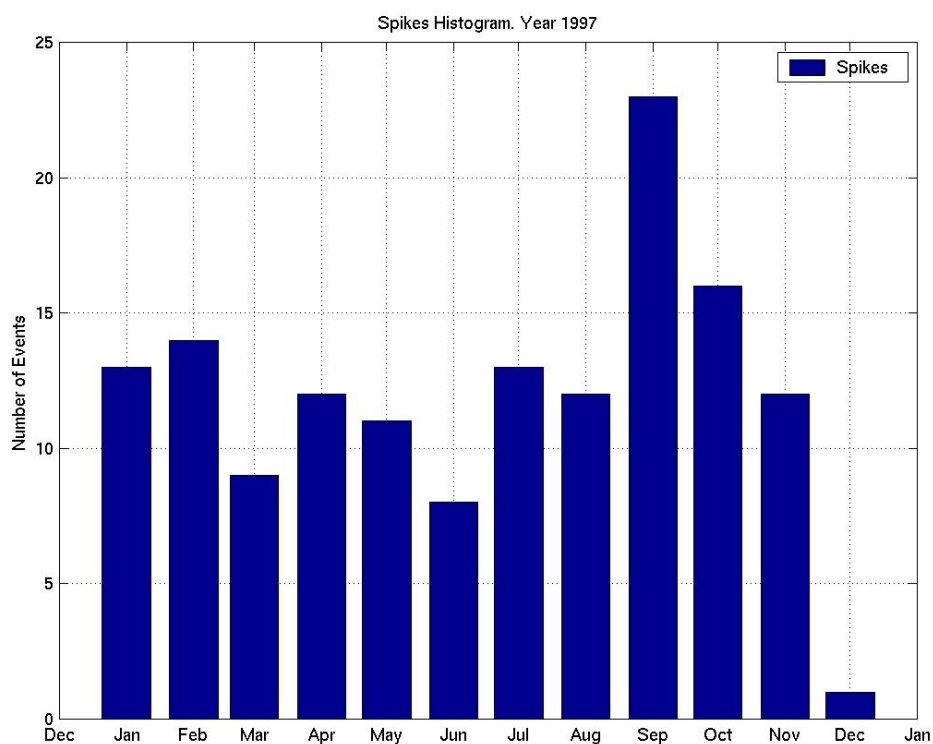


Figure 5.3.4c-2: Statistical Analysis III, Year 1997.

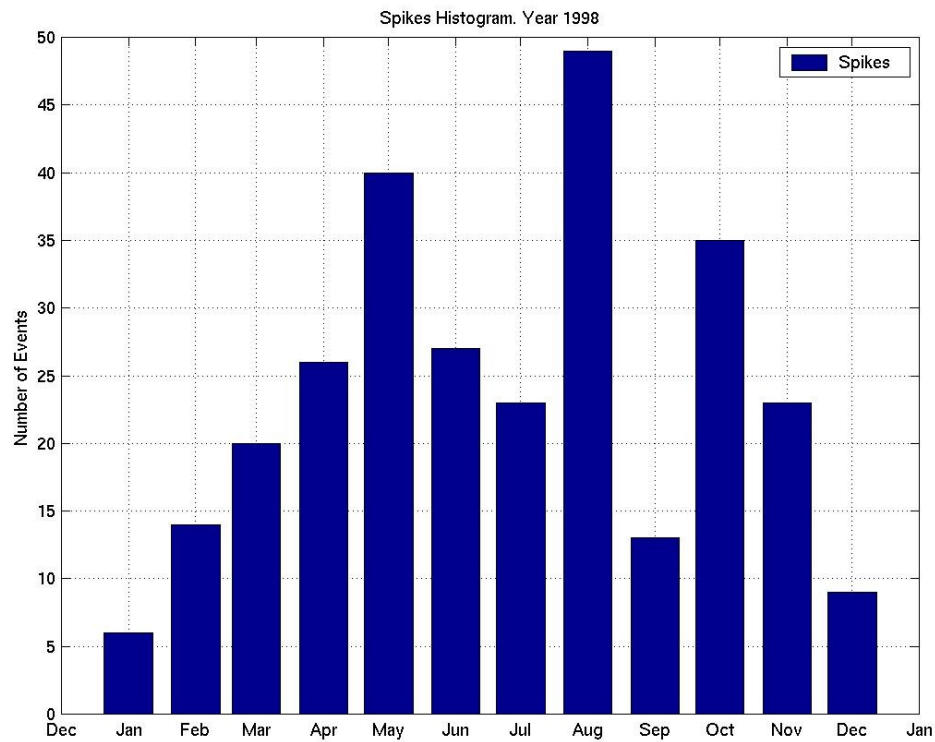


Figure 5.3.5c-2: Statistical Analysis III, Year 1998.

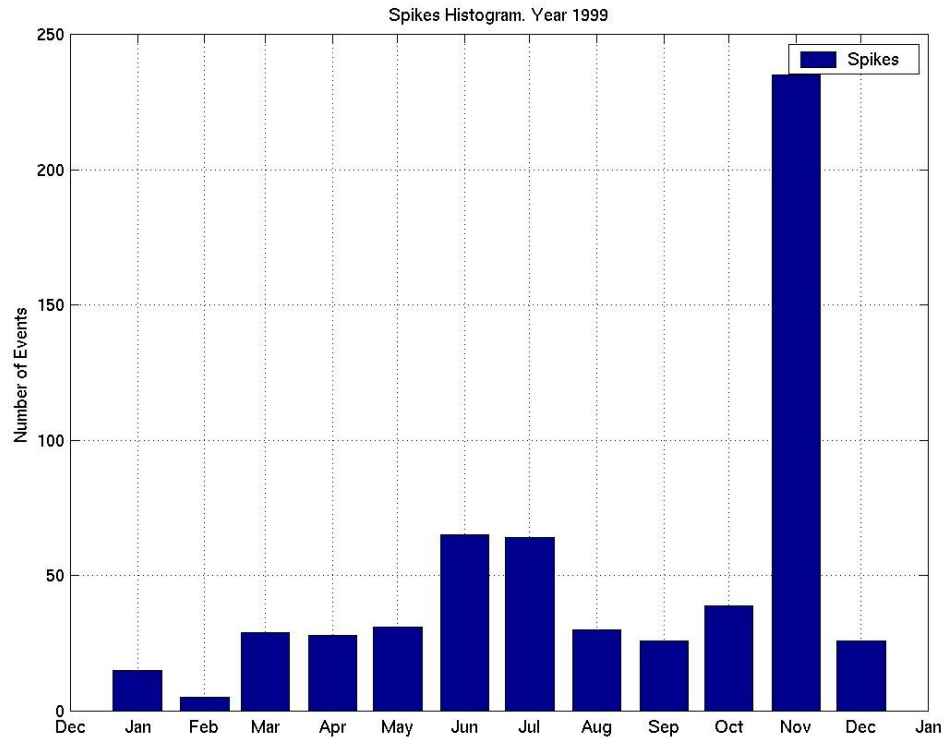


Figure 5.3.6c-2: Statistical Analysis III, Year 1999.

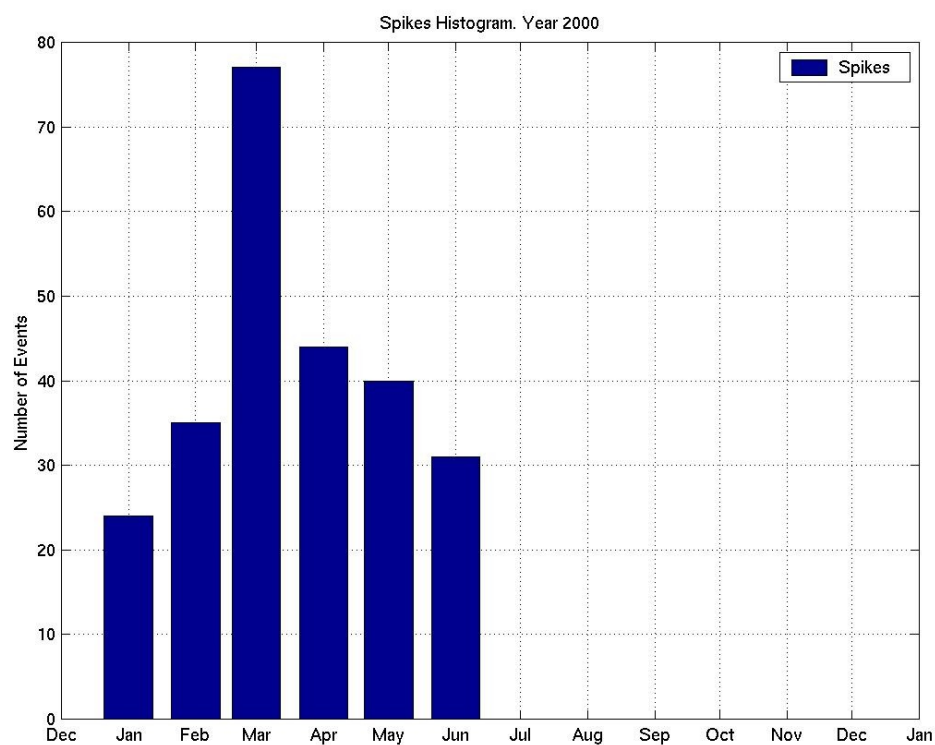


Figure 5.3.7c-2: Statistical Analysis III, Year 2000.

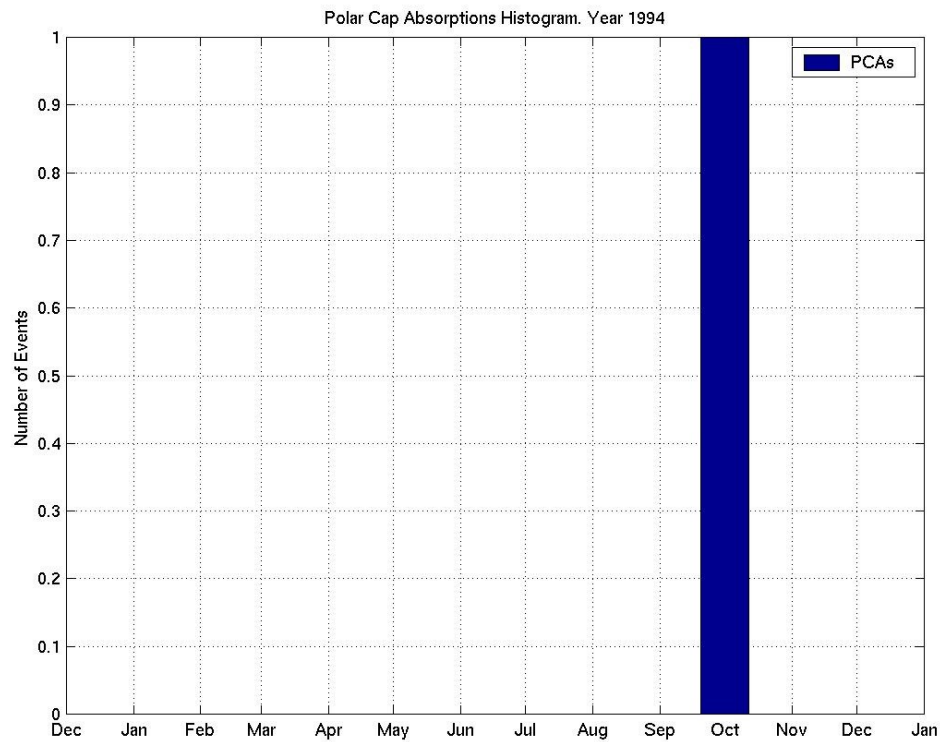


Figure 5.3.1c-3: Statistical Analysis III, Year 1994.

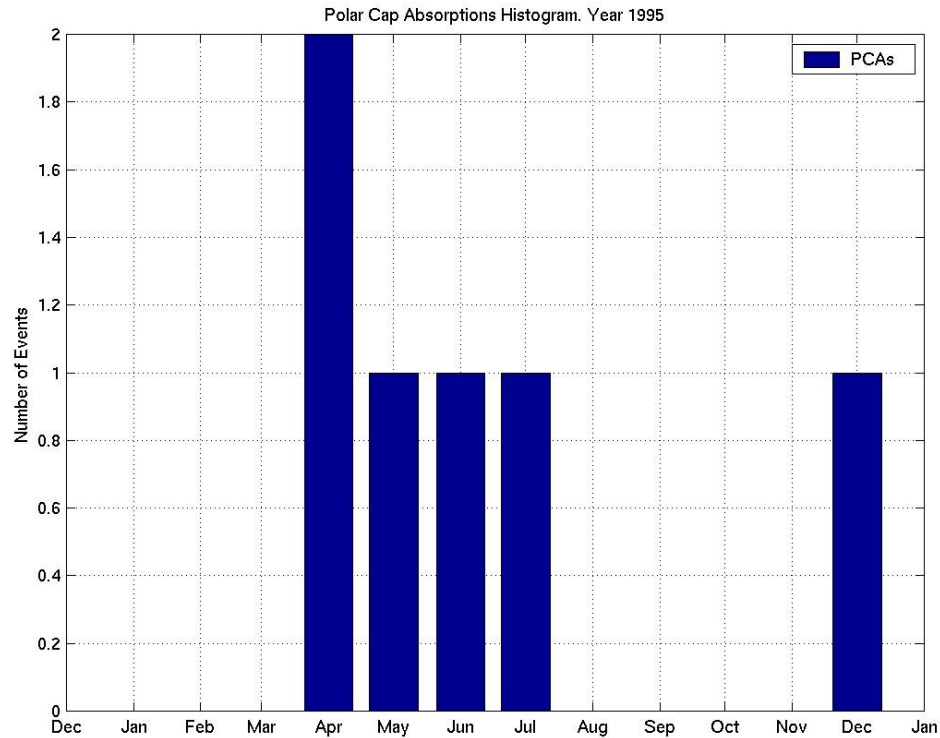


Figure 5.3.2c-3: Statistical Analysis III, Year 1995.

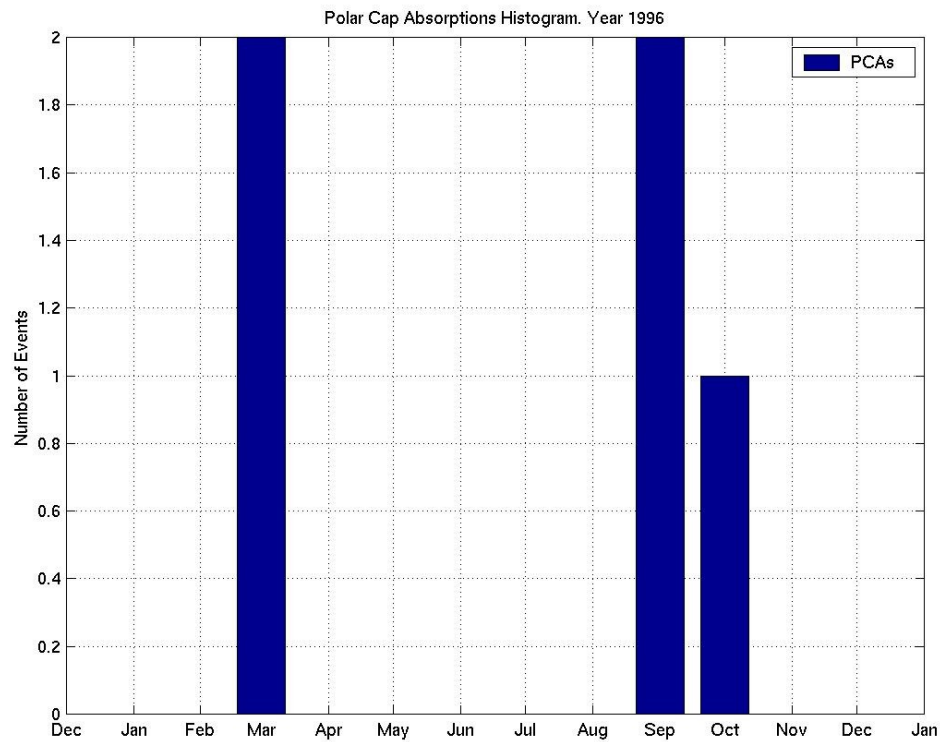


Figure 5.3.3c-3: Statistical Analysis III, Year 1996.

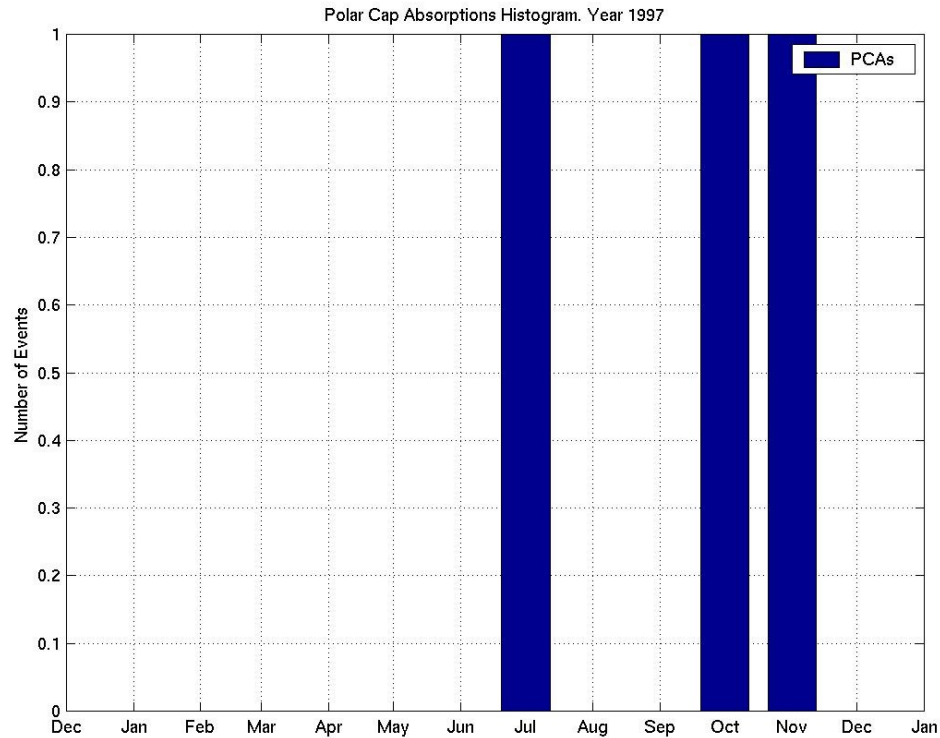


Figure 5.3.4c-3: Statistical Analysis III, Year 1997.

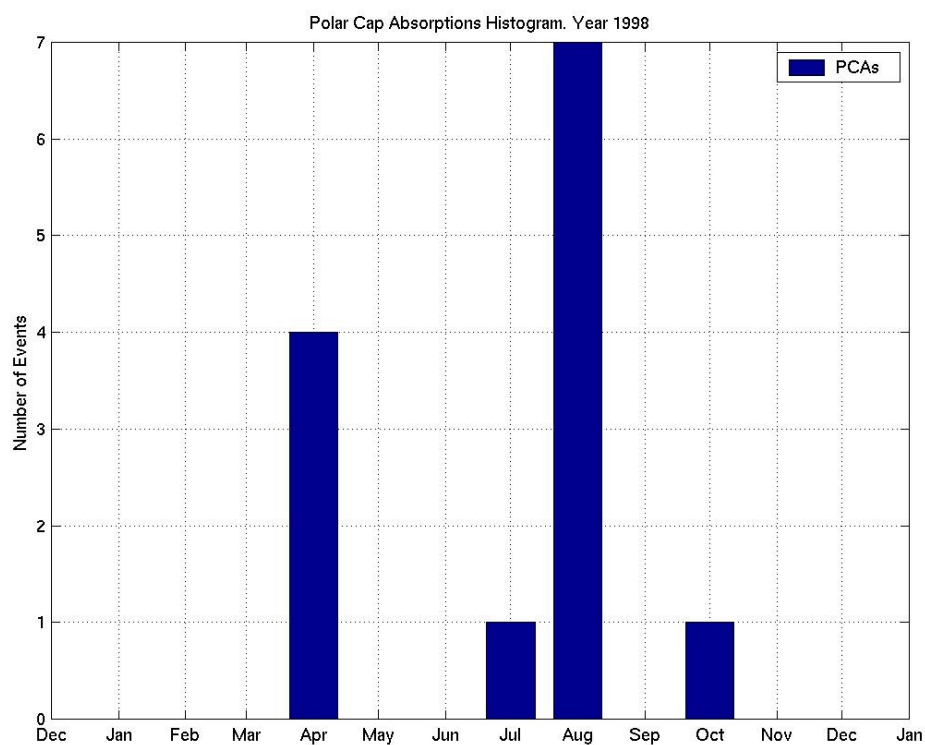


Figure 5.3.5c-3: Statistical Analysis III, Year 1998.

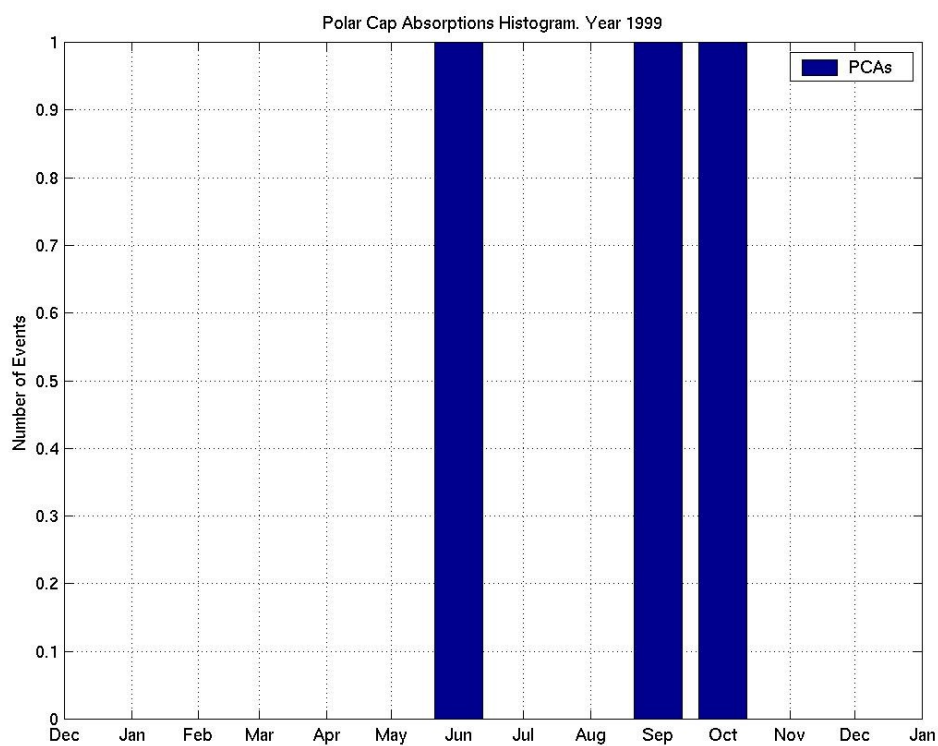


Figure 5.3.6c-3: Statistical Analysis III, Year 1999.

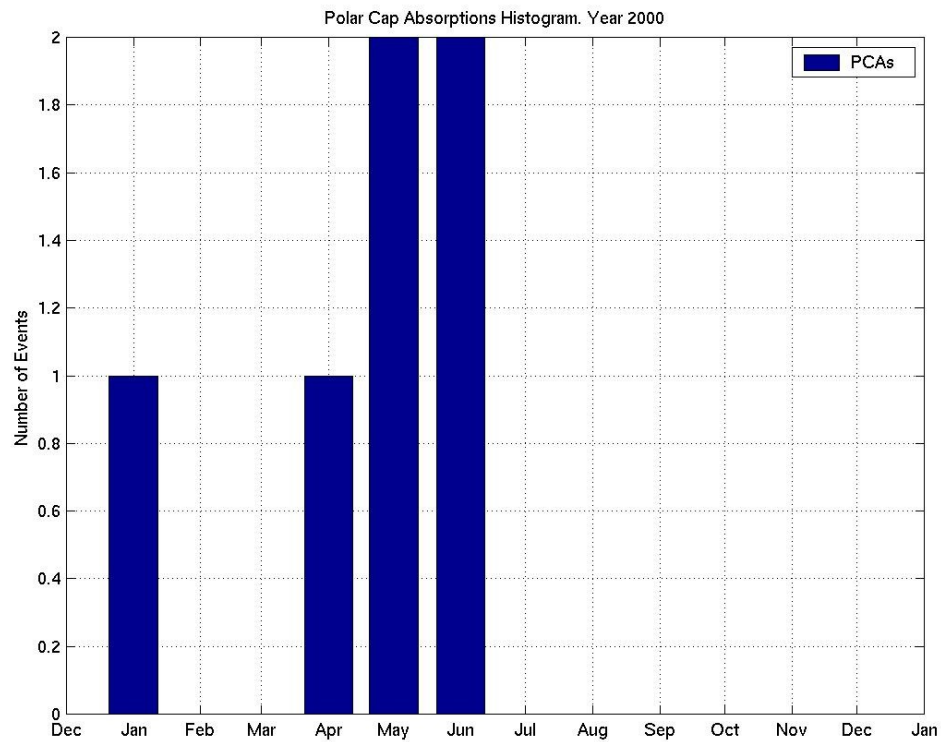


Figure 5.3.7c-3: Statistical Analysis III, Year 2000.

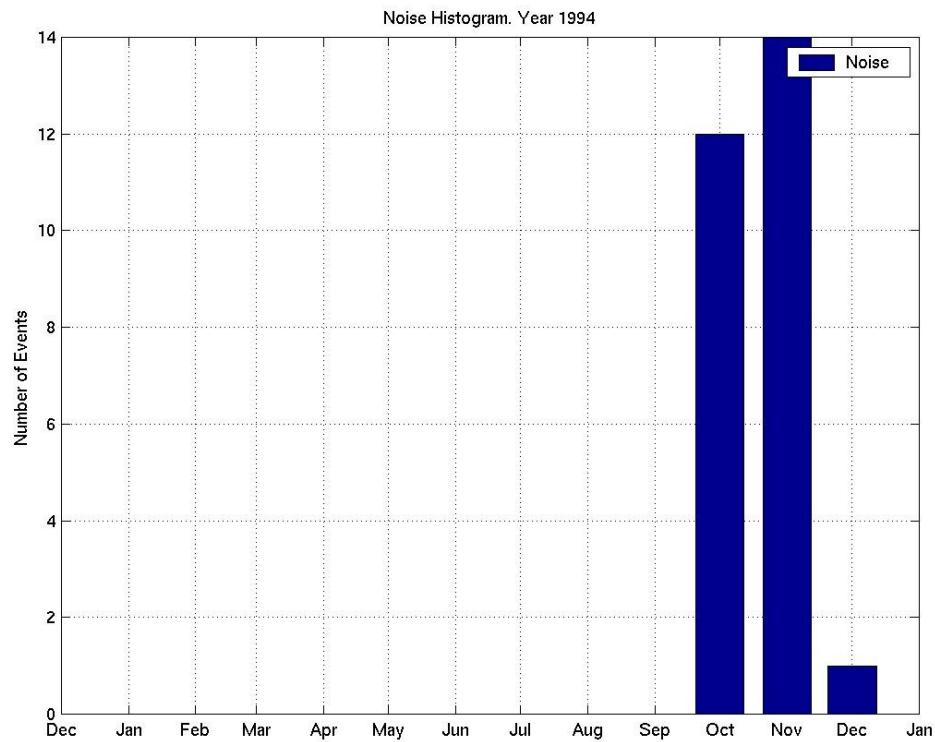


Figure 5.3.1c-4: Statistical Analysis III, Year 1994.

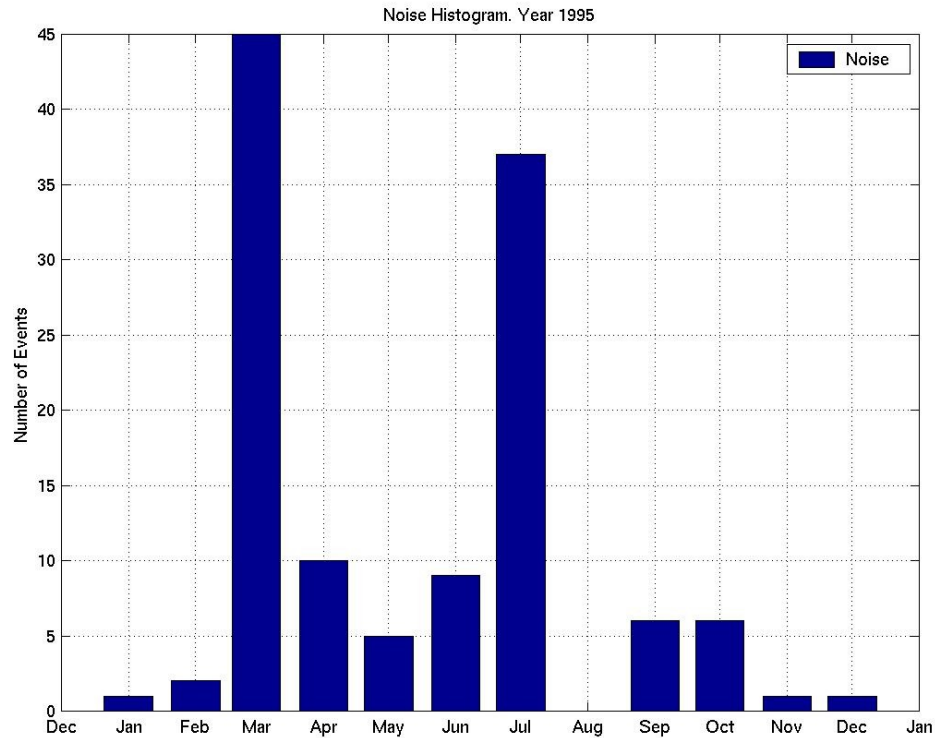


Figure 5.3.2c-4: Statistical Analysis III, Year 1995.

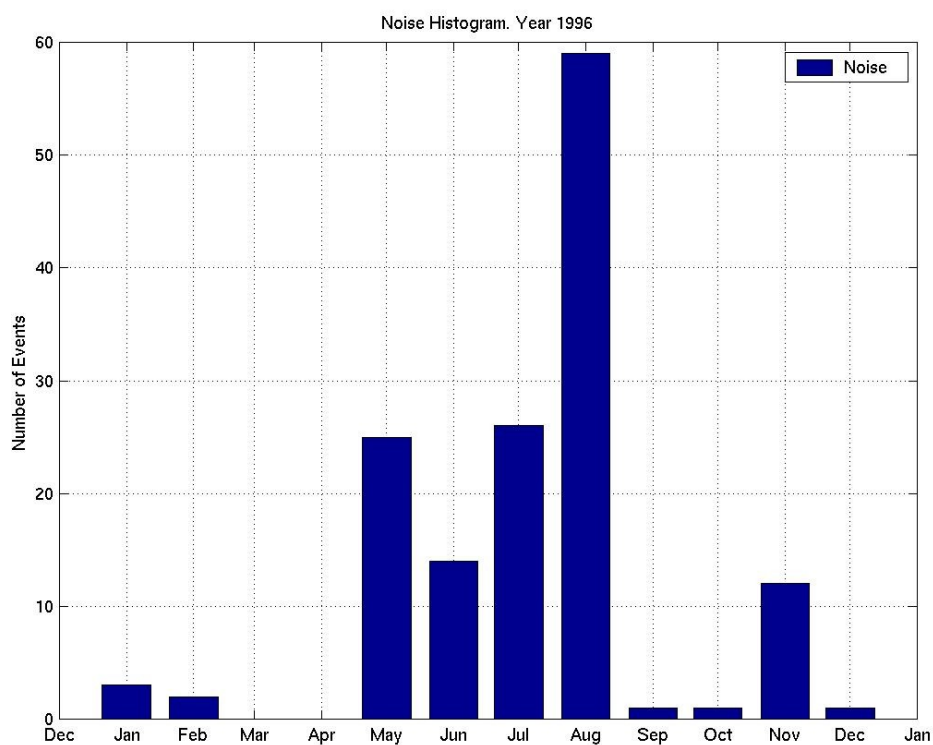


Figure 5.3.3c-4: Statistical Analysis III, Year 1996.

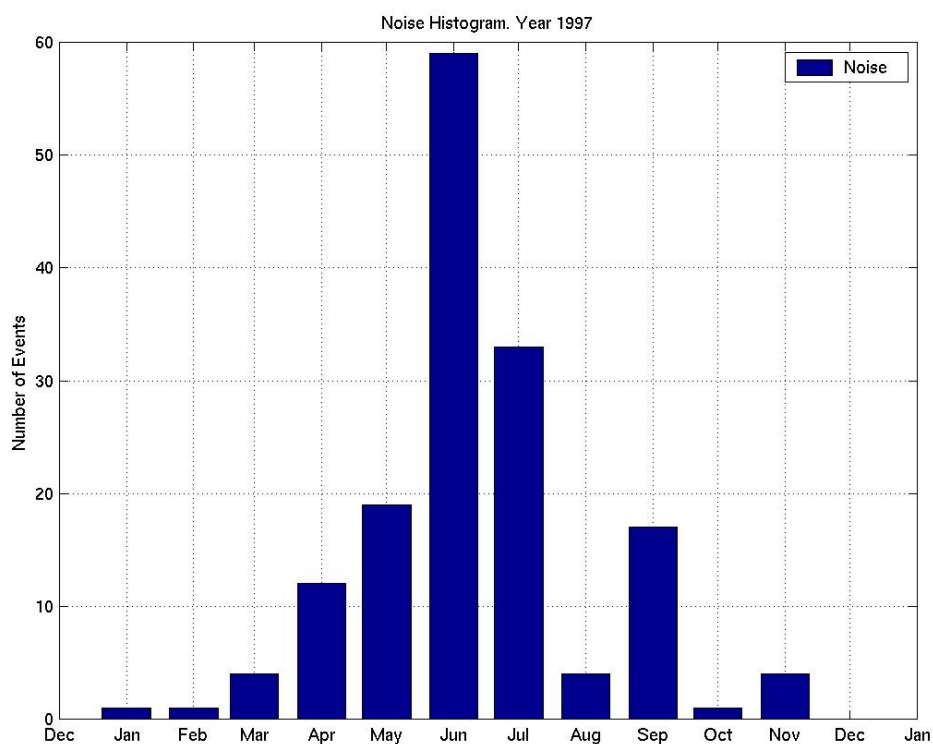


Figure 5.3.4c-4: Statistical Analysis III, Year 1997.

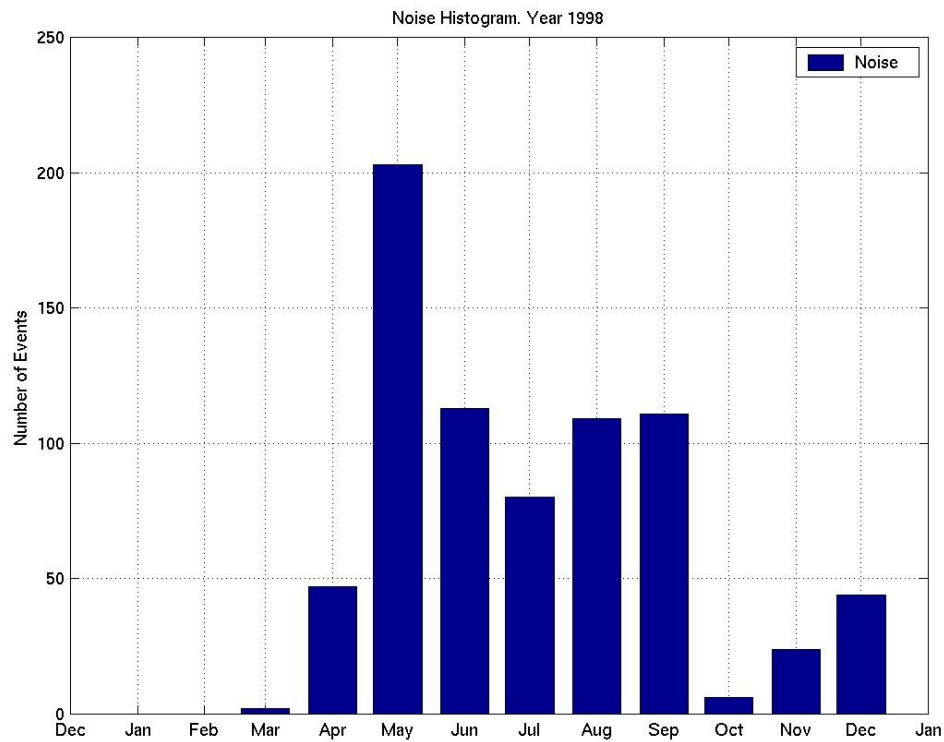


Figure 5.3.5c-4: Statistical Analysis III, Year 1998.

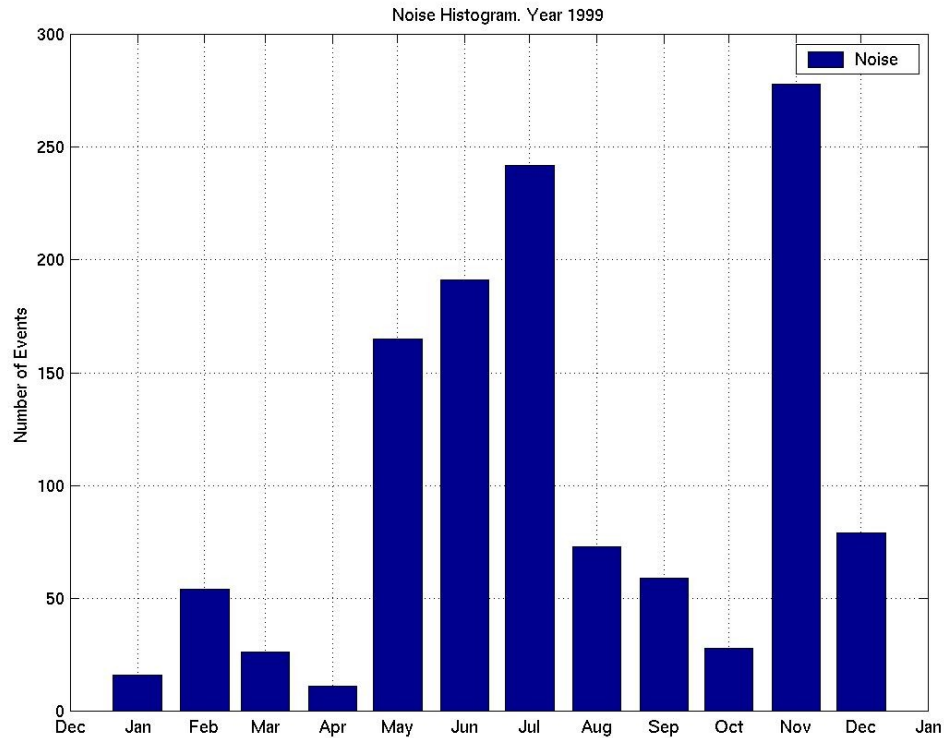


Figure 5.3.6c-4: Statistical Analysis III, Year 1999.

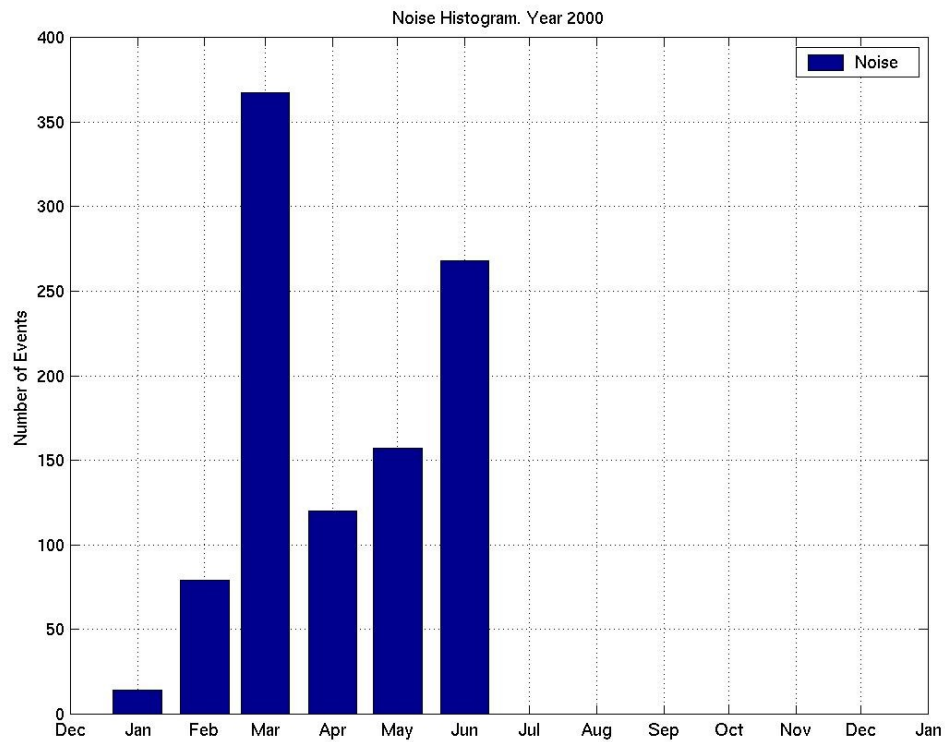


Figure 5.3.7c-4: Statistical Analysis III, Year 2000.

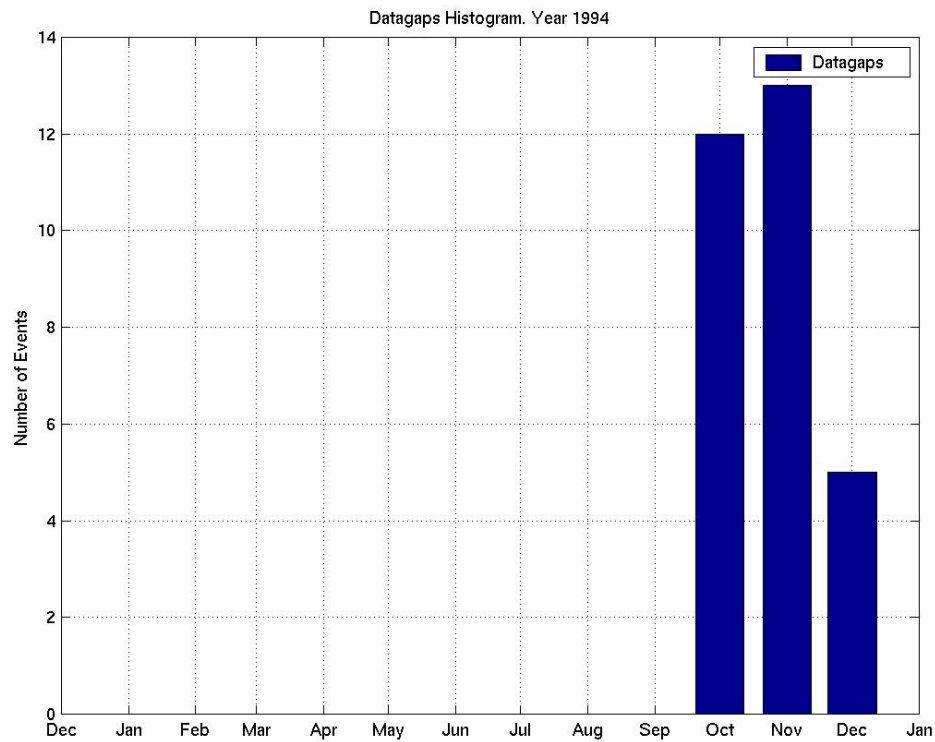


Figure 5.3.1c-5: Statistical Analysis III, Year 1994.

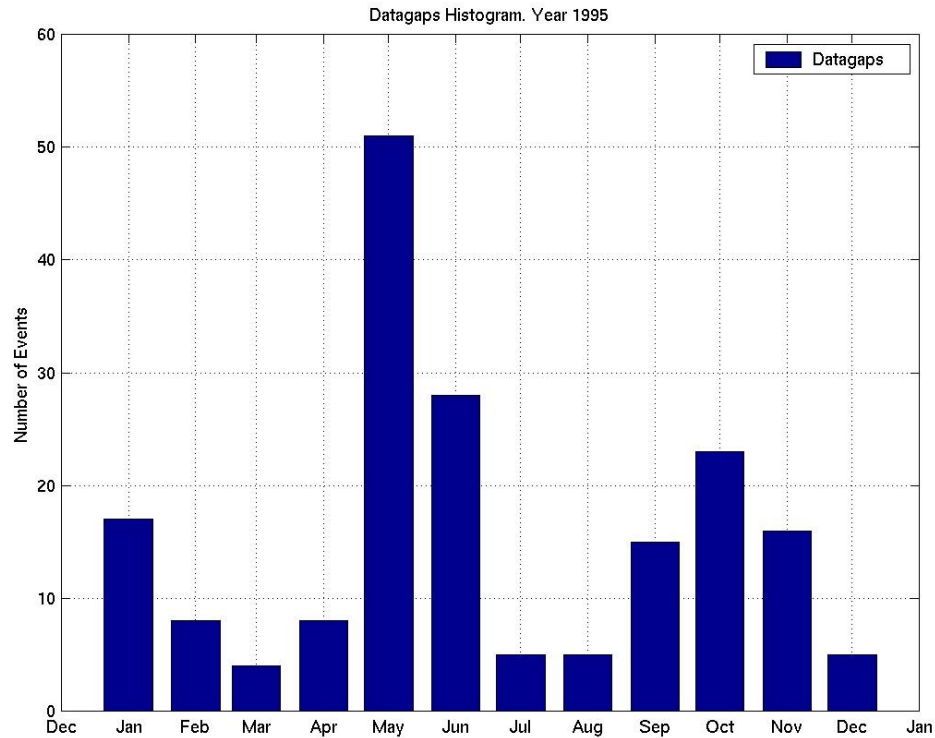


Figure 5.3.2c-5: Statistical Analysis III, Year 1995.

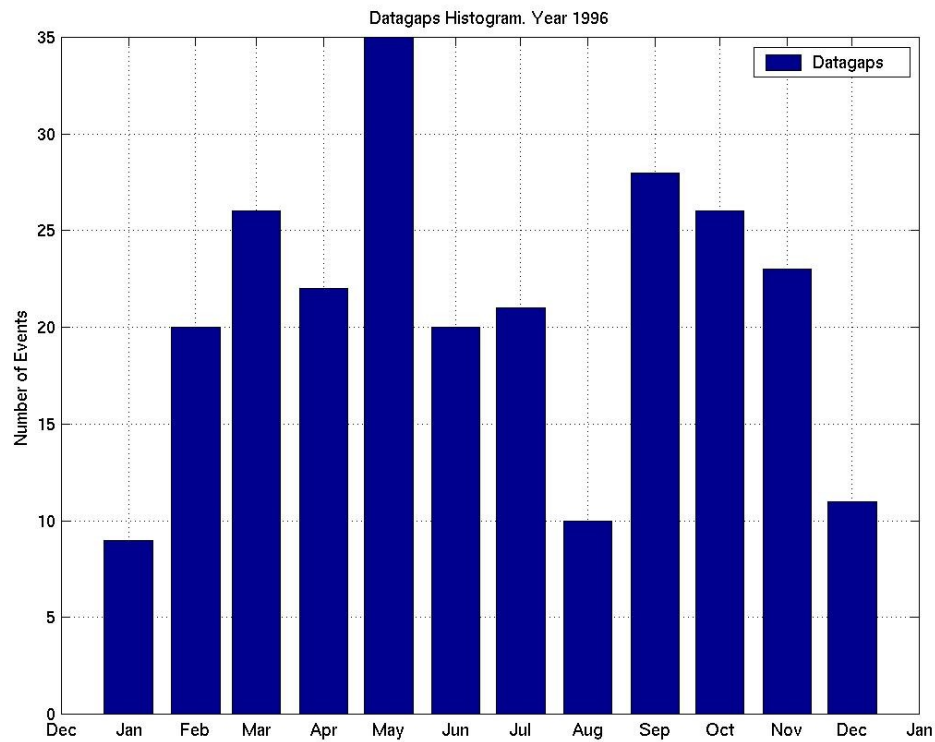


Figure 5.3.3c-5: Statistical Analysis III, Year 1996.

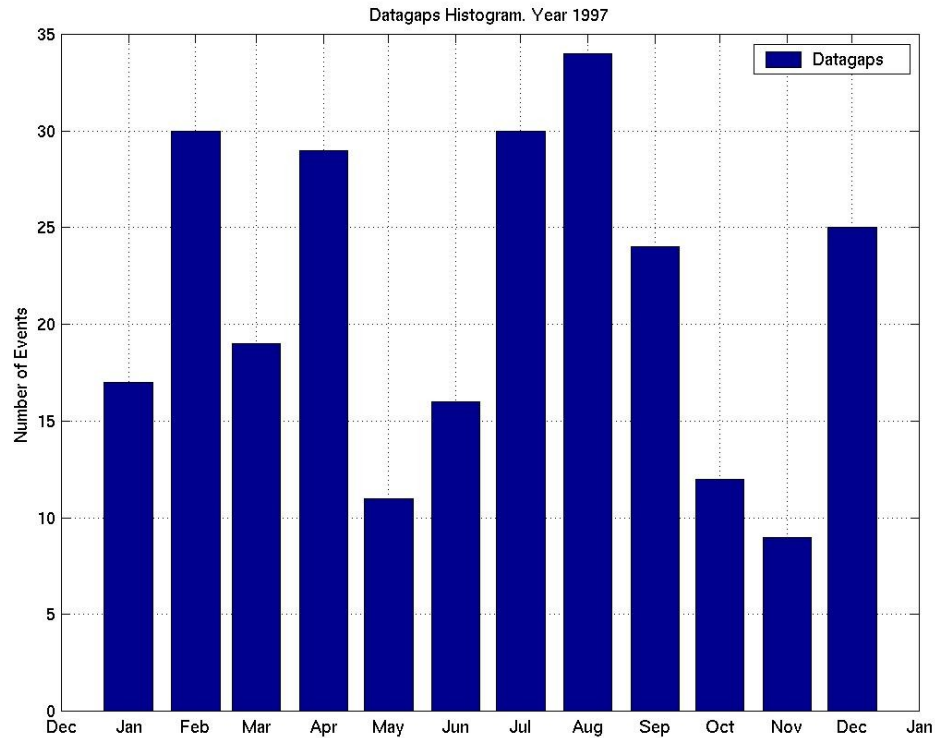


Figure 5.3.4c-5: Statistical Analysis III, Year 1997.

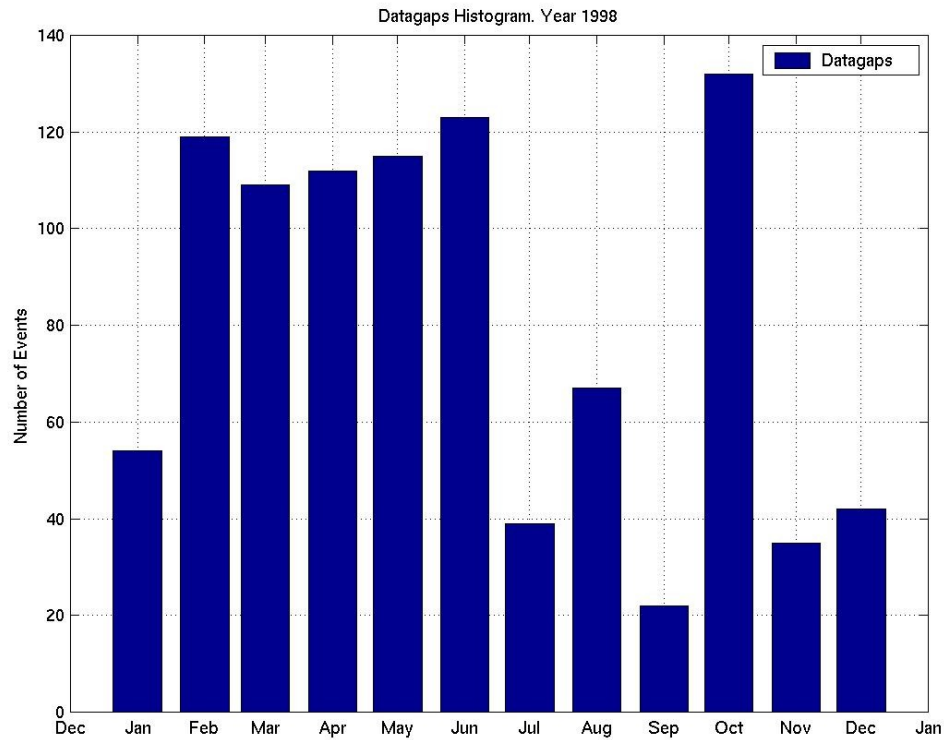


Figure 5.3.5c-5: Statistical Analysis III, Year 1998.

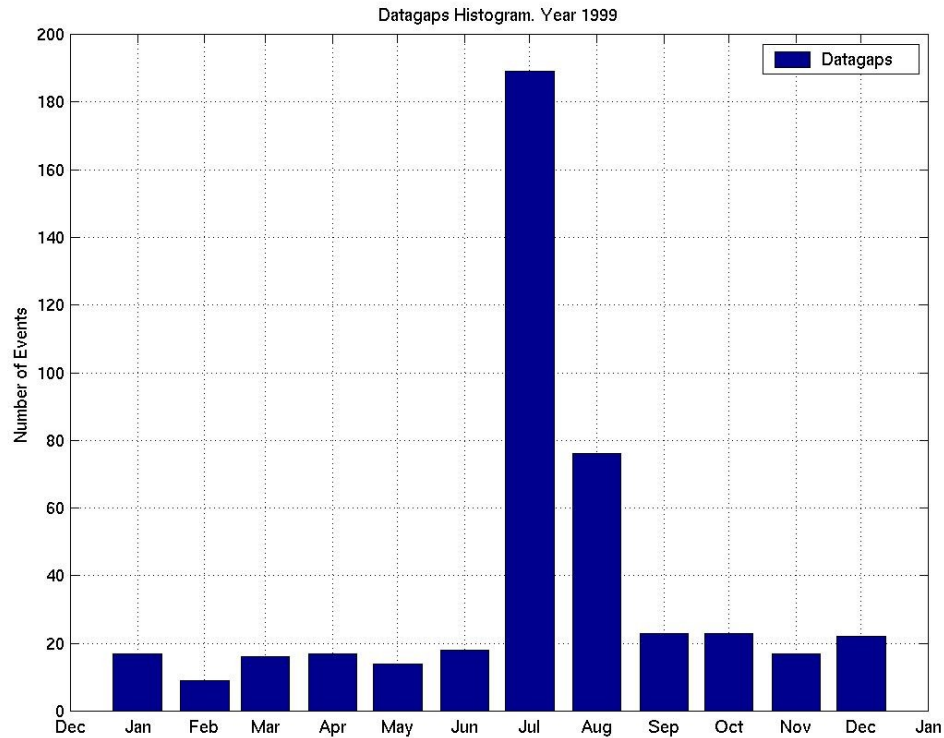


Figure 5.3.6c-5: Statistical Analysis III, Year 1999.

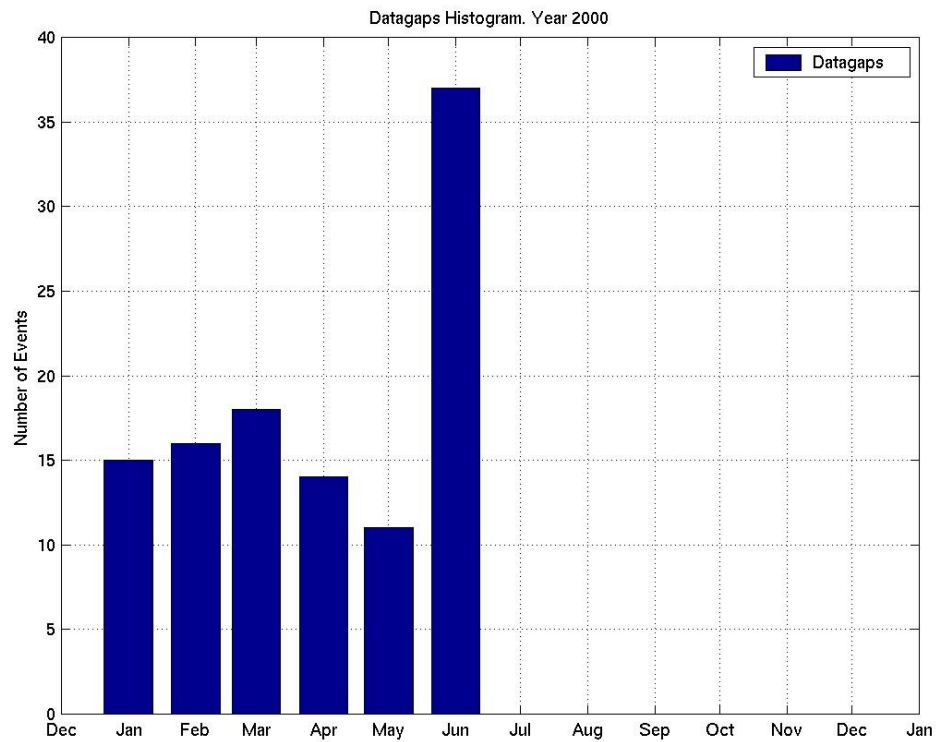


Figure 5.3.7c-5: Statistical Analysis III, Year 2000.